MATLAB® examples

MathWorks图书计划

MathWorks Fellow Jim Tung作序

科学与工程计算技术丛书

MATLAB OPTIMIZATION ALGORITHM

MATLAB优化算法

张 岩 吴水根◎编著

Zhang Yan Wu Shuigen

资深作者编著，图书质量更有保证

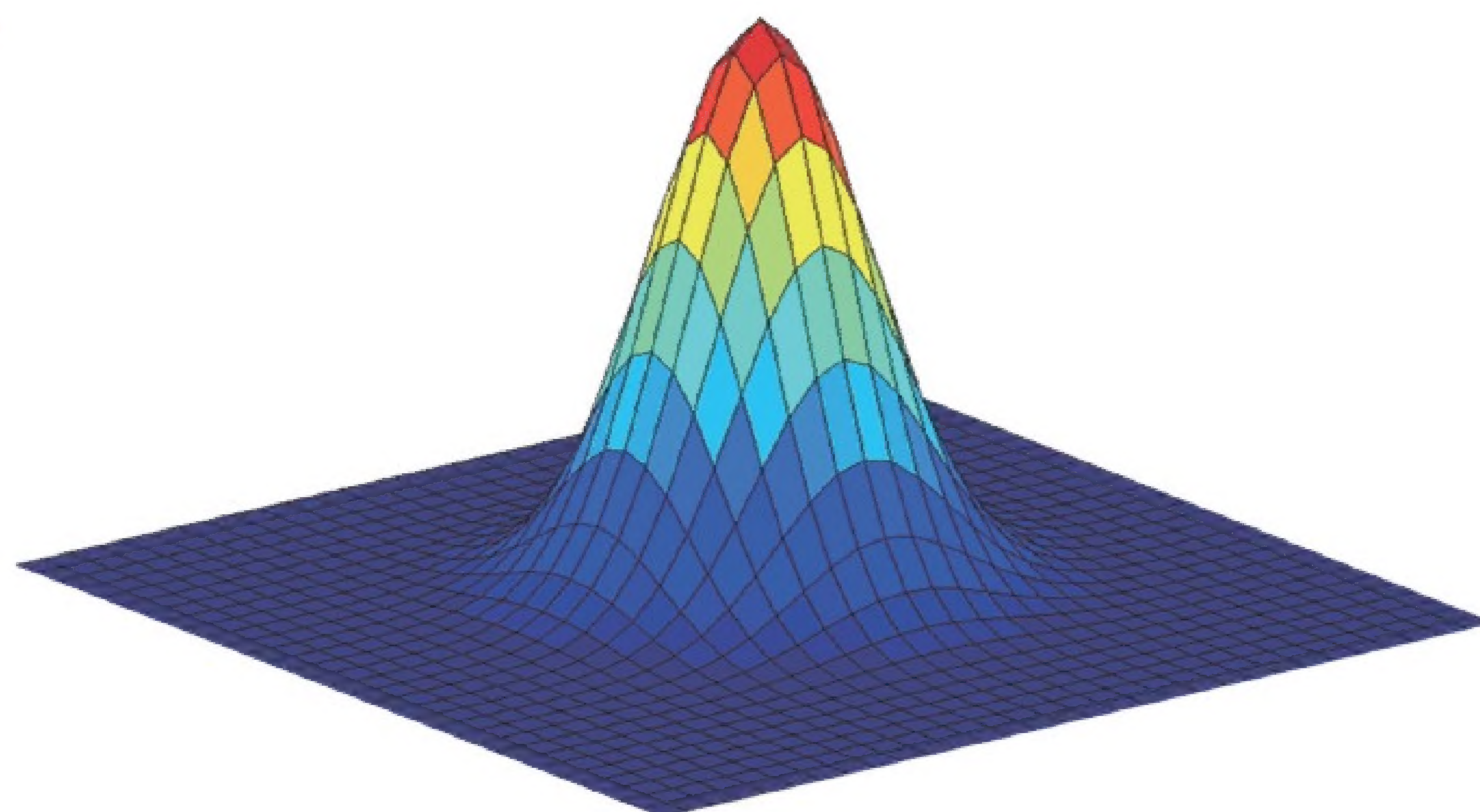
MATLAB资深工程师执笔，系统总结优化算法的实战经验

提供配套源码，便于读者动手实践

理论联系实践，本书提供源代码下载，方便读者学习使用

内含丰富实例，利于读者二次开发

提供了大量优化算法的典型实例，读者可以据此二次开发





清华大学出版社

科学与工程计算技术丛书

Optimization Algorithm with MATLAB

MATLAB 优化算法

(MATLAB R2016b 平台)

张岩　吴水根　编著

Zhang Yan　Wu Shuigen

清华大学出版社

北京

内 容 简 介

本书是一本简明的MATLAB优化算法综合性参考书，以MATLAB R2016b软件版本为基础，根据常用优化算法编写，包含多种优化算法的MATLAB应用方法，是读者掌握MATLAB在优化算法中应用的有力工具。

全书分为四个部分共18章，包括MATLAB应用基础、常规优化算法、智能优化算法和综合应用。第一部分从MATLAB基础知识开始，详细介绍编程和程序设计、二维绘图、三维绘图、GUI应用等内容；第二部分介绍MATLAB线性规划、非线性规划、无约束一维极值、无约束多维极值、约束优化方法、二次规划、多目标函数的优化方法等内容；第三部分介绍免疫优化算法及其MATLAB实现、粒子群优化算法的MATLAB实现、遗传优化算法的MATLAB实现、小波变换的MATLAB实现、神经网络的MATLAB实现等内容；第四部分主要介绍MATLAB在分形维数和经济金融最优化中的应用。在本书的最后，附录中还给出了MATLAB基本命令的介绍，便于读者查阅。

本书以MATLAB优化内容为主线，结合各种优化模型案例的讲解，各种MATLAB优化算法函数的说明，使读者易看懂、会应用。本书深入浅出，实例引导，讲解翔实，既可以作为高等院校数学建模和数学实验的参考教材，也可作为广大科研工程技术人员的参考用书。

图书在版编目(CIP)数据

MATLAB优化算法/张岩，吴水根编著. —北京：清华大学出版社，2017
(科学与工程计算技术丛书)
ISBN 978-7-302-47495-1

Ⅰ. ①M…　Ⅱ. ①张…　②吴…　Ⅲ. ①Matlab软件—应用—最优化算法　Ⅳ. ①O242.23-39

中国版本图书馆CIP数据核字(2017)第140417号

责任编辑：盛东亮
封面设计：李召霞
责任校对：梁　毅
责任印制：刘海龙

出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦A座　　**邮　　编：**100084
　　社 总 机：010-62770175　　**邮　　购：**010-62786544
　　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
　　课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　张：**30.25　　**字　　数：**734千字
版　　次：2017年11月第1版　　**印　　次：**2017年11月第1次印刷
印　　数：1～2000
定　　价：89.00元

产品编号：072487-01

序言

致力于加快工程技术和科学研究的步伐——这句话总结了 MathWorks 坚持超过三十年的使命。

在这期间，MathWorks 有幸见证了工程师和科学家使用 MATLAB 和 Simulink 在多个应用领域中的无数变革和突破：汽车行业的电气化和不断提高的自动化；日益精确的气象建模和预测；航空航天领域持续提高的性能和安全指标；由神经学家破解的大脑和身体奥秘；无线通信技术的普及；电力网络的可靠性，等等。

与此同时，MATLAB 和 Simulink 也帮助了无数大学生在工程技术和科学研究课程里学习关键的技术理念并应用于实际问题中，培养他们成为栋梁之材，更好地投入科研、教学以及工业应用中，指引他们致力于学习、探索先进的技术，融合并应用于创新实践中。

如今，工程技术和科研创新的步伐令人惊叹。创新进程以大量的数据为驱动，结合相应的计算硬件和用于提取信息的机器学习算法。软件和算法几乎无处不在——从孩子的玩具到家用设备，从机器人和制造体系到每一种运输方式——让这些系统更具功能性、灵活性、自主性。最重要的是，工程师和科学家推动了这些进程，他们洞悉问题，创造技术，设计革新系统。

为了支持创新的步伐，MATLAB 发展成为一个广泛而统一的计算技术平台，将成熟的技术方法（比如控制设计和信号处理）融入令人激动的新兴领域，例如深度学习、机器人、物联网开发等。对于现在的智能连接系统，Simulink 平台可以让您实现模拟系统，优化设计，并自动生成嵌入式代码。

“科学与工程计算技术丛书”系列主题反映了 MATLAB 和 Simulink 汇集的领域——大规模编程、机器学习、科学计算、机器人等。我们高兴地看到“科学与工程计算技术丛书”支持 MathWorks 一直以来追求的目标：助您加速工程技术和科学研究。

期待着您的创新！

Jim Tung

MathWorks Fellow

PREFACE

To Accelerate the Pace of Engineering and Science. These eight words have summarized the MathWorks mission for over 30 years.

In that time, it has been an honor and a humbling experience to see engineers and scientists using MATLAB and Simulink to create transformational breakthroughs in an amazingly diverse range of applications: the electrification and increasing autonomy of automobiles; the dramatically more accurate models and forecasts of our weather and climates; the increased performance and safety of aircraft; the insights from neuroscientists about how our brains and bodies work; the pervasiveness of wireless communications; the reliability of power grids; and much more.

At the same time, MATLAB and Simulink have helped countless students in engineering and science courses to learn key technical concepts and apply them to real-world problems, preparing them better for roles in research, teaching, and industry. They are also equipped to become lifelong learners, exploring for new techniques, combining them, and applying them in novel ways.

Today, the pace of innovation in engineering and science is astonishing. That pace is fueled by huge volumes of data, matched with computing hardware and machine-learning algorithms for extracting information from it. It is embodied by software and algorithms in almost every type of system-from children's toys to household appliances to robots and manufacturing systems to almost every form of transportation-making those systems more functional, flexible, and autonomous. Most important, that pace is driven by the engineers and scientists who gain the insights, create the technologies, and design the innovative systems.

To support today's pace of innovation, MATLAB has evolved into a broad and unifying technical computing platform, spanning well-established methods, such as control design and signal processing, with exciting newer areas, such as deep learning, robotics, and IoT development. For today's smart connected systems, Simulink is the platform that enables you to simulate those systems, optimize the design, and automatically generate the embedded code.

The topics in this book series reflect the broad set of areas that MATLAB and Simulink bring together: large-scale programming, machine learning, scientific computing, robotics, and more. We are delighted to collaborate on this series, in support of our ongoing goal: to enable you to accelerate the pace of your engineering and scientific work.

I look forward to the innovations that you will create!

Jim Tung
MathWorks Fellow

前言

MATLAB是美国MathWorks公司出品的商业数学软件，常用于算法开发、数据可视化、数据分析以及数值计算的高级技术计算语言和交互式环境。在优化算法中，MATLAB也有大量的应用。

优化算法有很多，关键是针对不同的优化问题，如可行解变量的取值（连续还是离散）、目标函数和约束条件的复杂程度（线性还是非线性）等，应用不同的算法。对于连续和线性等较简单的问题，可以选择一些经典算法，如梯度、Hessian矩阵、拉格朗日乘数、单纯形法、梯度下降法等。而对于更复杂的问题，则可考虑用一些智能优化算法，如遗传算法和蚁群算法。此外还包括模拟退火、禁忌搜索、粒子群算法等。

本书是利用MATLAB软件R2016b版本进行MATLAB优化算法应用的最新书籍。

1. 本书特点

（1）由浅入深，循序渐进：本书以有优化算法应用需求的读者为对象，首先从MATLAB应用基础知识讲起，接着详细讲解MATLAB求解各种优化问题，帮助读者尽快掌握MATLAB求解优化问题。

（2）步骤详尽，内容新颖：本书结合作者多年的MATLAB优化算法的使用经验与实际问题应用案例，将优化算法的分析及其MATLAB的实现方法和函数应用详细地讲解给读者。本书在讲解过程中步骤详尽、内容新颖，讲解过程辅以相应的图片，使读者在阅读时一目了然，从而快速掌握书中所讲内容。

（3）实例典型，轻松易学：书中多种优化算法求解案例，是掌握MATLAB优化算法和优化函数应用最好的方式。本书通过典型案例的求解，透彻、详尽地讲解了MATLAB在优化算法中的各种应用，即MATLAB优化函数的使用。

2. 本书内容

本书共18章，分为基础篇（MATLAB应用基础）、进阶篇（MATLAB常规优化算法）、高级篇（MATLAB智能优化算法）、综合应用篇四部分，帮助初、中级读者快速掌握MATLAB优化算法应用。本书基于MATLAB R2016b版，详细讲解MATLAB优化算法的基础知识和经典案例。具体内容如下：

第一部分为MATLAB应用基础部分。主要介绍MATLAB各种基础运算、编程和程序设计、二维绘图、三维绘图、GUI应用等内容。其中每章内容安排如下：

第1章：MATLAB基础知识　　第2章：MATLAB编程

第3章：MATLAB绘图　　第4章：GUI应用

第二部分为MATLAB常规优化算法部分。主要介绍MATLAB线性规划、非线性规划、无约束一维极值、无约束多维极值、约束优化方法、二次规划、多目标函数的优化方法等内容。其中每章内容如下：

前言

第 5 章：MATLAB 线性规划　　第 6 章：MATLAB 非线性规划
第 7 章：无约束一维极值　　第 8 章：无约束多维极值
第 9 章：约束优化方法　　第 10 章：二次规划
第 11 章：多目标函数的优化方法

第三部分为 MATLAB 智能优化算法部分。主要介绍免疫优化算法及其 MATLAB 实现、粒子群优化算法的 MATLAB 实现、遗传优化算法的 MATLAB 实现、小波变换的 MATLAB 实现、神经网络的 MATLAB 实现等内容。其中每章内容如下：

第 12 章：免疫优化算法及其实现　　第 13 章：粒子群优化算法的实现
第 14 章：遗传优化算法的实现　　第 15 章：小波变换的实现
第 16 章：神经网络的实现

第四部分为 MATLAB 综合应用部分。主要介绍分形维数应用与 MATLAB 实现、经济金融最优化 MATLAB 应用等内容。其中每章内容如下：

第 17 章：分形维数应用与实现　　第 18 章：经济金融最优化应用

3. 读者对象

本书适合 MALTAB 初学者和期望掌握 MATLAB 优化应用的读者。具体说明如下：

★相关从业人员　　★初学数学建模的技术人员
★大中专院校的教师和在校生　　★相关培训机构的教师和学员
★参加工作实习的“菜鸟”　　★广大科研工作人员

4. 读者服务

为了方便解决本书疑难问题，读者朋友在学习过程中如果遇到与本书有关的技术问题，可以发邮件到邮箱 caxart@126.com，或者访问博客 http://blog.sina.com.cn/caxart，编者会尽快给予解答。

另外本书所涉及的素材文件(程序代码)已经上传到本书提供的博客中，读者可以自行下载。

5. 本书作者

本书主要由张岩、吴水根编著。此外，付文利、王广、沈再阳、林晓阳、任艳芳、唐家鹏、孙国强、高飞等也参与了本书部分内容的编写工作，在此表示感谢。

虽然作者在本书的编写过程中力求叙述准确、完善，但由于水平有限，书中欠妥之处在所难免，希望读者和同仁能够及时指出，共同促进本书质量的提高。

最后再次希望本书能为读者的学习和工作提供帮助！

编　者
2017.7.1

目录

第一部分　MATLAB 应用基础

目录

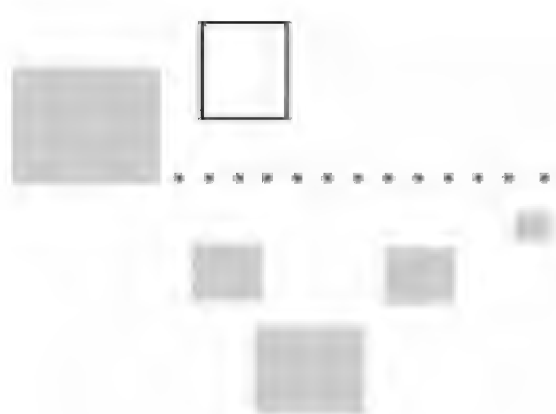
第二部分 MATLAB常规优化算法

目录

第三部分 MATLAB 智能优化算法

目录

第四部分 MATLAB 综合应用



第一部分
MATLAB应用基础

第1章 MATLAB基础知识

MATLAB是目前在国际上被广泛接受和使用的科学与工程计算软件，在优化算法中有广泛应用。本章主要介绍MATLAB优化的基础知识，包括基本概念、向量、数组、矩阵、字符串、符号、关系运算和逻辑运算等内容。

学习目标：

(1) 了解MATLAB基本概念；

(2) 掌握MATLAB中的向量、数组、矩阵等运算；

(3) 熟练掌握MATLAB数据类型间的转换。

1.1 基本概念

20世纪70年代中后期，曾在密歇根大学、斯坦福大学和新墨西哥大学担任数学与计算机科学教授的Cleve Moler博士，为讲授矩阵理论和数值分析课程的需要，他和同事用Fortran语言编写了两个子程序库EISPACK和LINPACK，这便是构思和开发MATLAB的起点。MATLAB一词是对matrix laboratory(矩阵实验室)的缩写，由此可看出MATLAB与矩阵计算的渊源。

数据类型、常量与变量是MATLAB语言入门时必须引入的一些基本概念，MATLAB虽是一个集多种功能于一体的集成软件，但就其语言部分而言，这些概念不可缺少。

1.1.1 数据类型概述

数据作为计算机处理的对象，在程序语言中可分为多种类型，MATLAB作为一种可编程的语言当然也不例外。MATLAB的主要数据类型如图1-1所示。

MATLAB数值型数据划分成整型和浮点型的用意和C语言有所不同。MATLAB的整型数据主要为图像处理等特殊的应用问题提供数据类型，以便节省空间或提高运行速度。对一般数值运算，绝大多数情况是采用双精度浮点型的数据。

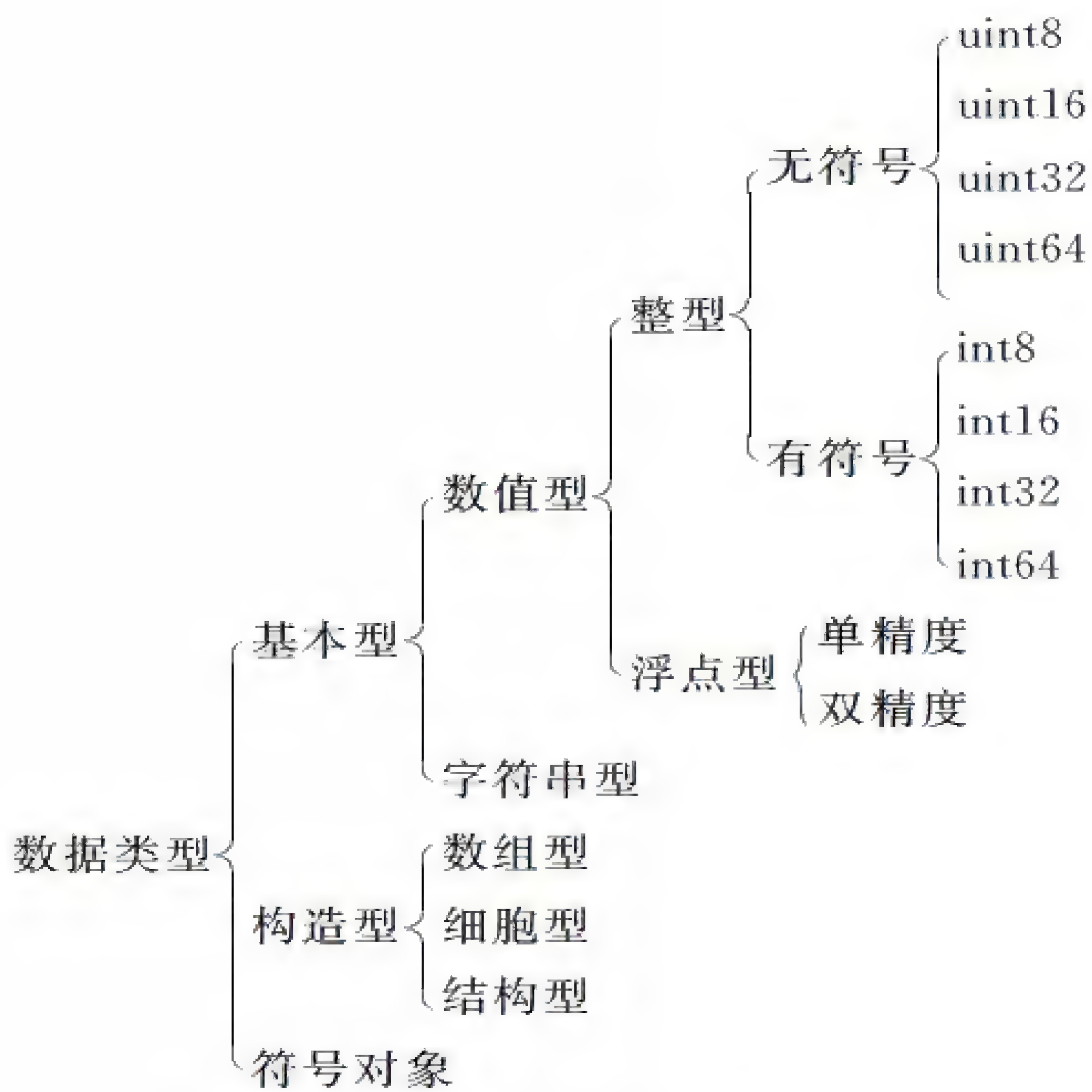


图 1-1 MATLAB 主要数据类型

MATLAB 的构造型数据基本上与 C++的构造型数据相衔接，但它的数组却有更加广泛的含义不同于一般语言的运算方法。

符号对象是 MATLAB 所特有的一类为符号运算而设置的数据类型。严格地说，它不是某一类型的数据，它可以是数组、矩阵、字符等多种形式及其组合，但它在 MATLAB 的工作区中的确是另立的一种数据类型。

在使用中，MATLAB 数据类型有一个突出的特点：在对不同数据类型的变量引用时，一般不用事先对变量的数据类型进行定义或说明，系统会依据变量被赋值的类型自动进行类型识别，这在高级语言中是极有特色的。

这样处理的优势是，在书写程序时可以随时引入新的变量而不用担心出现什么问题，这的确给应用带来很大方便。但缺点是有失严谨，会给搜索和确定一个符号是否为变量名带来更多的时间开销。

1.1.2 整数类型

MATLAB 中提供了 8 种内置的整数类型，表 1-1 中列出了它们各自存储占用位数、能表示数值的方位和转换函数。

表 1-1 MATLAB 中的整数类型

整数类型	数值范围	转换函数
有符号 8 位整数	$-2^7 \sim 2^7-1$	int8
无符号 8 位整数	$0 \sim 2^8-1$	uint8
有符号 16 位整数	$-2^{15} \sim 2^{15}-1$	int16
无符号 16 位整数	$0 \sim 2^{16}-1$	uint16
有符号 32 位整数	$-2^{31} \sim 2^{31}-1$	int32

续表

整数类型	数值范围	转换函数
无符号32位整数	$0\sim2^{32}-1$	uint32
有符号64位整数	$-2^{63}\sim2^{63}-1$	int64
无符号64位整数	$0\sim2^{64}-1$	uint64

不同的整数类型所占用的位数不同，因此所能表示的数值范围不同。在实际应用中，应该根据需要的数据范围选择合适的整数类型。有符号的整数类型拿出一位来表示正负，因此表示的数据范围和相应的无符号整数类型不同。

由于 MATLAB 中数值的默认存储类型是双精度浮点类型，因此，必须通过表1-2中列出的转换函数将双精度浮点数值转换成指定的整数类型。

表1-2　MATLAB中的取整函数

函　　数	说　　明	举　　例
round(a)	向最接近的整数取整 小数部分是0.5时向绝对值大的方向取整	round(4.3)结果为4 round(4.5)结果为5
fix(a)	向0方向取整	fix(4.3)结果为4 fix(4.5)结果为4
floor(a)	向不大于a的最接近整数取整	floor(4.3)结果为4 floor(4.5)结果为4
ceil(a)	向不小于a的最接近整数取整	ceil(4.3)结果为5 ceil (4.5)结果为5

在转换中，MATLAB 默认将待转换数值转换为最接近的整数，若小数部分为0.5，那么 MATLAB 转换后的结果是绝对值较大的那个整数。另外，应用这些转换函数也可以将其他类型转换成指定的整数类型。

【例1-1】 通过转换函数创建整数类型。

解：在 MATLAB 命令行窗口输入：

```
>> x = 105;y = 105.49;z = 105.5;
>> xx = int16(x) % 把 double 型变量 x 强制转换成 int16 型
xx =
    105
>> yy = int32(y)
yy =
    105
>> zz = int32(z)
zz =
    106
```

MATLAB 中还有多种取整函数，可以用不同的策略把浮点小数转换成整数，如表1-2所示。

数据类型参与的数学运算与 MATLAB 中默认的双精度浮点运算不同。当两种相同的整数类型进行运算时，结果仍然是这种整数类型；当一个整数类型数值与一个双精

度浮点类型数值进行数学运算时，计算结果是这种整数类型，取整采用默认的四舍五入方式。需要注意的是，两种不同的整数类型之间不能进行数学运算，除非提前进行强制转换。

【例 1-2】 整数类型数值参与的运算。

解： 在 MATLAB 命令行窗口输入：

```
>> x = uint32(367.2) * uint32(20.3)
x =
        7340
>> y = uint32(24.321) * 359.63
y =
        8631
>> z = uint32(24.321) * uint16(359.63)
Error using .*
Integers can only be combined with integers of the same class, or scalar doubles.
>> whos
  Name      Size        Bytes       Class       Attributes
  x         1x1         4           uint32
  y         1x1         4           uint32
```

前面表 1-1 中已经介绍了不同的整数类型能够表示的数值范围不同。数学运算中，运算结果超出相应的整数类型能够表示的范围时，就会出现溢出错误，运算结果被置为该整数类型能够表示的最大值或最小值。

MATLAB 提供了 intwarning 函数可以设置是否显示这种转换或计算过程中出现的溢出，即非正常转换的错误，有兴趣的读者可以参考 MATLAB 的联机帮助。

1.1.3 浮点数类型

MATLAB 中提供了单精度浮点数类型和双精度浮点数类型，它们在存储位宽、各位用处、表示的数值范围、数值精度等方面都不同，如表 1-3 所示。

表 1-3 MALTAB 中单精度浮点数和双精度浮点数的比较

浮点类型	存储位宽	各数据位的用处	数值范围	转换函数
双精度	64	0～51 位表示小数部分 52～62 位表示指数部分 63 位表示符号(0 位正，1 位负)	−1.79769e+308～2.22507e−308 2.22507e−308～1.79769e+308	double
单精度	32	0～22 位表示小数部分 23～30 位表示指数部分 31 位表示符号(0 位正，1 位负)	−3.40282e+038～−1.17549e−038 −1.17549e−038～3.40282e+038	single

从表 1-3 可以看出，存储单精度浮点类型所用的位数少，因此内存占用上开支小。但从各数据位的用处来看，单精度浮点数能够表示的数值范围和数值精度都比双精度小。

和创建整数类型数值一样，创建浮点数类型也可以通过转换函数来实现。当然，MATLAB 中默认的数值类型是双精度浮点类型。

【例 1-3】 浮点数转换函数的应用。

解：在 MATLAB 命令行窗口输入：

```
>> x = 5.4
x =
    5.4000
>> y = single(x)  % 把 double 型的变量强制转换为 single
y =
    5.4000
>> z = uint32(87563);
>> zz = double(z)
zz =
    87563
>> whos
  Name      Size      Bytes   Class      Attributes
  x         1x1       8       double
  y         1x1       4       single
  z         1x1       4       uint32
  zz        1x1       8       double
```

双精度浮点数参与运算时，返回值的类型依赖于参与运算中的其他数据类型。双精度浮点数与逻辑型、字符型进行运算时，返回结果为双精度浮点类型；而与整数型进行运算时返回结果为相应的整数类型；与单精度浮点型运算时返回单精度浮点型。单精度浮点型与逻辑型、字符型和任何浮点型进行运算时，返回结果都是单精度浮点型。

注意：单精度浮点型不能和整数型进行算术运算。

【例 1-4】 浮点型参与的运算。

解：在 MATLAB 命令行窗口输入：

```
>> x = uint32(240);y = single(32.345);z = 12.356;
>> xy = x * y
Error using .*
Integers can only be combined with integers of the same class, or
scalar doubles.
>> xz = x * z
xz =
        2965
>> whos
  Name      Size      Bytes   Class      Attributes
  x         1x1       4       uint32
  xz        1x1       4       uint32
  y         1x1       4       single
  z         1x1       8       double
```

从表 1-3 可以看出，浮点数只占用一定的存储位宽，其中只有有限位分别用来存储指数部分和小数部分。因此，浮点类型能表示的实际数值是有限的，而且是离散的。任何两个最接近的浮点数之间都有一个很微小的间隙，而所有处在这个间隙中的值都只能用这两个最接近的浮点数中的一个来表示。MATLAB 中提供了 eps 函数，可以获取一个

数值和它最接近的浮点数之间的间隙大小。

1.1.4 常量与变量

1. 常量

常量是程序语句中取不变值的那些量。如表达式 y=0.618 * x,其中就包含一个 0.618 这样的数值常数,它便是一数值常量。而另一表达式 s='Tomorrow and Tomorrow'中,单引号内的英文字符串“Tomorrow and Tomorrow”则是字符串常量。

在 MATLAB 中,有一类常量是由系统默认给定一个符号来表示的。例如 pi,它代表圆周率 π 这个常数,即 3.1415926…类似于 C 语言中的符号常量。这些常量如表 1-4 所列,有时又称为系统预定义的变量。

表 1-4 MATLAB 特殊常量

常量符号	常量含义
i 或 j	虚数单位,定义为 $i^2=j^2=-1$
Inf 或 inf	正无穷大,由零作除数引入此常量
NaN	不定值,表示非数值量,产生于 0/0、∞/∞、0×∞等运算
pi	圆周率 π 的双精度表示
eps	容差变量,当某量的绝对值小于 eps 时,可以认为此量为零,即为浮点数的最小分辨率,PC 上此值为 2^{-52}
Realmin 或 realmin	最小浮点数,2^{-1022}
Realmax 或 realmax	最大浮点数,2^{1023}

MATLAB 中存在一些预定义的特殊变量,称为常量。常用的 MATLAB 常量如表 1-5 所示。

表 1-5 MATLAB 常用常量

常量	说明	常量	说明
i,j	虚数单位,定义为 $\sqrt{-1}$	eps	浮点运算的相对精度
pi	圆周率	realmax	最大的正实数
Inf	无穷大	realmin	最小的正实数
NaN	不定值(0/0)	ans	默认变量名

在 MATLAB 中,定义变量时应避免与常量名相同,以免改变常数的值,为计算带来不便。

【例 1-5】 在 MATLAB 命令行窗口输入:

```
>> eps
>> pi
```

运行程序输出结果如下:

```
>> eps
ans =
    2.2204e-16
>> pi
ans =
    3.1416
```

2. 变量

变量是在程序运行中其值可以改变的量，变量由变量名来表示。在 MATLAB 中变量名的命名有自己的规则，可以归纳成如下几条：

（1）变量名必须以字母开头，且只能由字母、数字或者下画线 3 类符号组成，不能含有空格和标点符号（如“()”“,”“.”“%”）等。

（2）变量名区分字母的大小写。例如，“a”和“A”是不同的变量。

（3）变量名不能超过 63 个字符，第 63 个字符后的字符被忽略，对于 MATLAB 6.5 版以前的变量名不能超过 31 个字符。

（4）关键字（如 if、while 等）不能作为变量名。

（5）最好不要用表 1-4 中的特殊常量符号作变量名。

常见的错误命名如 f(x)、y'、y"、A_2 等。

1.1.5 数组、矩阵、向量和标量

数组、矩阵、向量和标量是 MATLAB 运算中涉及的一组基本运算量。它们各自的特点及相互间的关系可以描述如下：

（1）数组不是一个数学量，而是一个用于高级语言程序设计的概念。如果数组元素按一维线性方式组织在一起，那么称其为一维数组，一维数组的数学原型是向量。

如果数组元素分行、列排成一个二维平面表格，那么称其为二维数组，二维数组的数学原型是矩阵。如果元素在排成二维数组的基础上，再将多个行、列数分别相同的二维数组叠成一本立体表格，便形成三维数组。以此类推下去，便有了多维数组的概念。

在 MATLAB 中，数组的用法与一般高级语言不同。它不借助于循环，而是直接采用运算符，有自己独立的运算符和运算法则。

（2）矩阵是一个数学概念，一般高级语言并未引入其作为基本的运算量，但 MATLAB 是个例外。一般高级语言是不认可将两个矩阵视为两个简单变量而直接进行加减乘除的，要完成矩阵的四则运算必须借助于循环结构。

当 MATLAB 将矩阵引入作为基本运算量后，上述局面改变了。MATLAB 不仅实现了矩阵的简单加减乘除运算，而且许多与矩阵相关的其他运算也因此大大简化了。

（3）向量是一个数学量，一般高级语言中也未引入，它可视为矩阵的特例。从 MATLAB 的工作区窗口可以查看到：一个 n 维的行向量是一个 $1\times n$ 阶的矩阵，而列向量则当成 $n\times 1$ 阶的矩阵。

（4）标量也是一个数学概念，但在 MATLAB 中，一方面可将其视为一般高级语言的

简单变量来处理，另一方面又可把它当成 1×1 阶的矩阵，这一看法与矩阵作为 MATLAB 的基本运算量是一致的。

(5) 在 MATLAB 中，二维数组和矩阵其实是数据结构形式相同的两种运算量。二维数组和矩阵的表示、建立、存储没有根本区别，区别只在它们的运算符和运算法则不同。

例如，向命令行窗口中输入 a=[1 2;3 4]这个量，实际上它有两种可能的角色：矩阵 a 或二维数组 a。这就是说，单从形式上是不能完全区分矩阵和数组的，必须再看它使用什么运算符与其他量之间进行运算。

(6) 数组的维和向量的维是两个完全不同的概念。数组的维是从数组元素排列后所形成的空间结构去定义的，即线性结构是一维，平面结构是二维，立体结构是三维，当然还有四维以至多维。向量的维相当于一维数组中的元素个数。

1.1.6 字符型数据

类似于其他高级语言，MATLAB 的字符和字符串运算也相当强大。在 MATLAB 中，字符串可以用单引号(')进行赋值，字符串的每个字符(含空格)都是字符数组的一个元素。MATLAB 还包含很多字符串相关操作函数，具体见表 1-6。

表 1-6 字符串操作函数

函数名	说明	函数名	说明
char	生成字符数组	strsplit	在指定的分隔符处拆分字符串
strcat	水平连接字符串	strtok	寻找字符串中记号
strvcat	垂直连接字符串	upper	转换字符串为大写
strcmp	比较字符串	lower	转换字符串为小写
strncmp	比较字符串的前 n 个字符	blanks	生成空字符串
strfind	在其他字符串中寻找此字符串	deblank	移去字符串内空格
strrep	以其他字符串代替此字符串		

【例 1-6】 在 MATLAB 命令窗口输入：

```
clear all
clc
syms a b
y=2*a+1;
```

运行程序输出结果如下：

```
>> y
y =
    2*a + 1
```

字符串的相减运算操作如下：

```
y1 = a + 2;
y2 = y - y1
```

运行程序输出结果如下：

```
y2 =
    a - 1
```

字符串的相加运算操作如下：

```
y3 = y + y1
```

运行程序输出结果如下：

```
y3 =
    3 * a + 3
```

字符串的相乘运算操作如下：

```
y4 = y * y1
```

运行程序输出结果如下：

```
y4 =
    (2 * a + 1) * (a + 2)
```

字符串的相除运算操作如下：

```
y5 = y/y1
```

运行程序输出结果如下：

```
y5 =
    (2 * a + 1)/(a + 2)
```

1.1.7 运算符

MATLAB 运算符可分为三大类，它们是算术运算符、关系运算符和逻辑运算符。下面分别给出它们的运算符和运算法则。

1. 算术运算符

算术运算因所处理的对象不同，分为矩阵和数组算术运算两类。表 1-7 给出的是矩阵算术运算的符号、名称、示例和使用说明。表 1-8 给出的是数组算术运算的运算符号、名称、示例和使用说明。

表 1-7　矩阵算术运算符

运算符	名　称	示　例	法则或使用说明
+	加	C=A+B	矩阵加法法则，即 C(i,j)=A(i,j)+B(i,j)
-	减	C=A-B	矩阵减法法则，即 C(i,j)=A(i,j)-B(i,j)
*	乘	C=A*B	矩阵乘法法则
/	右除	C=A/B	定义为线性方程组 X*B=A 的解，即 $C=A/B=A*B^{-1}$
\	左除	C=A\B	定义为线性方程组 A*X=B 的解，即 $C=A\backslash B=A^{-1}*B$
^	乘幂	C=A^B	A、B 其中一个为标量时有定义
'	共轭转置	B=A'	B 是 A 的共轭转置矩阵

表 1-8　数组算术运算符

运算符	名　称	示　例	法则或使用说明
.*	数组乘	C=A.*B	C(i,j)=A(i,j)*B(i,j)
./	数组右除	C=A./B	C(i,j)=A(i,j)/B(i,j)
.\	数组左除	C=A.\B	C(i,j)=B(i,j)/A(i,j)
.^	数组乘幂	C=A.^B	C(i,j)=A(i,j)^B(i,j)
.'	转置	A.'	将数组的行摆放成列，复数元素不做共轭

针对表 1-7 和表 1-8 需要说明以下几点：

(1) 矩阵的加、减、乘运算是严格按矩阵运算法则定义的，而矩阵的除法虽和矩阵求逆有关系，但却分了左、右除，因此不是完全等价的。乘幂运算更是将标量幂扩展到矩阵作为幂指数。总的来说，MATLAB 接受了线性代数已有的矩阵运算规则，但又不仅止于此。

(2) 表 1-8 中并未定义数组的加减法，是因为矩阵的加减法与数组的加减法相同，所以未做重复定义。

(3) 不论是加减乘除，还是乘幂，数组的运算都是元素间的运算，即对应下标元素一对一的运算。

(4) 多维数组的运算法则，可依元素按下标一一对应参与运算的原则将表 1-8 推广。

2. 关系运算符

MATLAB 关系运算符列在表 1-9 中。

表 1-9　关系运算符

运算符	名称	示例	法则或使用说明
<	小于	A<B	(1) A、B 都是标量，结果是或为 1(真)或为 0(假)的标量 (2) A、B 若一个为标量，另一个为数组，标量将与数组各元素逐一比较，结果为与运算数组行列相同的数组，其中各元素取值 1 或 0 (3) A、B 均为数组时，必须行、列数分别相同，A 与 B 各对应元素相比较，结果为与 A 或 B 行列相同的数组，其中各元素取值 1 或 0 (4) == 和 ~= 运算对参与比较的量同时比较实部和虚部，其他运算只比较实部
<=	小于等于	A<=B	
>	大于	A>B	
>=	大于等于	A>=B	
==	恒等于	A==B	
~=	不等于	A~=B	

需要明确指出的是，MATLAB的关系运算虽可看成矩阵的关系运算，但严格地讲，把关系运算定义在数组基础之上更为合理。因为从表1-9所列法则不难发现，关系运算是元素一对一的运算结果。数组的关系运算向下可兼容一般高级语言中所定义的标量关系运算。

3. 逻辑运算符

逻辑运算在MATLAB中同样需要，为此MATLAB定义了自己的逻辑运算符，并设定了相应的逻辑运算法则，如表1-10所示。

表1-10 逻辑运算符

<table>
<tr><th>运算符</th><th>名称</th><th>示例</th><th>法则或使用说明</th></tr>
<tr><td>&</td><td>与</td><td>A&B</td><td rowspan="5">(1) A、B都为标量，结果是或为1(真)或为0(假)的标量
(2) A、B若一个为标量，另一个为数组，标量将与数组各元素逐一做逻辑运算，结果为与运算数组行列相同的数组，其中各元素取值或1或0
(3) A、B均为数组时，必须行、列数分别相同，A与B各对应元素做逻辑运算，结果为与A或B行列相同的数组，其中各元素取值或1或0
(4) 先决与、先决或是只针对标量的运算</td></tr>
<tr><td>|</td><td>或</td><td>A|B</td></tr>
<tr><td>~</td><td>非</td><td>~A</td></tr>
<tr><td>&&</td><td>先决与</td><td>A&&B</td></tr>
<tr><td>||</td><td>先决或</td><td>A||B</td></tr>
</table>

同样地，MATLAB的逻辑运算也是定义在数组的基础之上，向下可兼容一般高级语言中所定义的标量逻辑运算。为提高运算速度，MATLAB还定义了针对标量的先决与和先决或运算。

先决与运算是当该运算符的左边为1(真)时，才继续与该符号右边的量做逻辑运算。先决或运算是当运算符的左边为1(真)时，就不需要继续与该符号右边的量做逻辑运算，而立即得出该逻辑运算结果为1(真)；否则，就要继续与该符号右边的量运算。

4. 运算符的优先级

和其他高级语言一样，当用多个运算符和运算量写出一个MATLAB表达式时，运算符的优先次序是一个必须明确的问题。表1-11列出了运算符的优先次序。

表1-11 MATLAB运算符的优先次序

<table>
<tr><th>优先次序</th><th>运　算　符</th></tr>
<tr><td>最高</td><td>'(转置共轭)、^(矩阵乘幂)、.'(转置)、.^(数组乘幂)</td></tr>
<tr><td></td><td>~(逻辑非)</td></tr>
<tr><td></td><td>*、/(右除)、\(左除)、.*(数组乘)、./(数组右除)、.\(数组左除)</td></tr>
<tr><td></td><td>+、-、:(冒号运算)</td></tr>
<tr><td></td><td><、<=、>、>=、==(恒等于)、~=(不等于)</td></tr>
<tr><td></td><td>&(逻辑与)</td></tr>
<tr><td></td><td>|(逻辑或)</td></tr>
<tr><td></td><td>&&(先决与)</td></tr>
<tr><td>最低</td><td>||(先决或)</td></tr>
</table>

MATLAB 运算符的优先次序在表 1-11 中依照从上到下的顺序，分别由高到低。而表中同一行的各运算符具有相同的优先级，而在同一级别中又遵循有括号先括号运算的原则。

1.1.8 复数

复数是对实数的扩展，每一个复数包括实部和虚部两部分。MATLAB 中默认用字符 i 或者 j 表示虚部。创建复数可以直接输入或者利用 complex 函数。

MATLAB 中还有多种对复数操作的函数，如表 1-12 所示。

表 1-12 MATLAB 中复数相关运算函数

函　数	说　明	函　数	说　明
real(z)	返回复数 z 的实部	imag(z)	返回复数 z 的虚部
abs(z)	返回复数 z 的幅度	angle(z)	返回复数 z 的幅角
conj(z)	返回复数 z 的共轭复数	complex(a,b)	以 a 为实部，b 为虚部创建复数

【例 1-7】 复数的创建和运算。

解：在 MATLAB 命令行窗口输入：

```
>> a = 2 + 3i
a =
   2.0000 + 3.0000i
>> x = rand(3) * 5;
>> y = rand(3) * - 8;
>> z = complex(x,y) % 用 somplex 函数创建以 x 为实部,y 为虚部的复数
z =
   4.0736 - 7.7191i   4.5669 - 7.6573i   1.3925 - 1.1351i
   4.5290 - 1.2609i   3.1618 - 3.8830i   2.7344 - 3.3741i
   0.6349 - 7.7647i   0.4877 - 6.4022i   4.7875 - 7.3259i
>> whos
  Name      Size      Bytes  Class     Attributes
  a         1x1          16  double    complex
  x         3x3          72  double
  y         3x3          72  double
  z         3x3         144  double    complex
```

1.1.9 无穷量和非数值量

MATLAB 中用 Inf 和 -Inf 分别代表正无穷和负无穷，用 NaN 表示非数值的值。正负无穷的产生一般是由于 0 做了分母或者运算溢出，产生了超出双精度浮点数数值范围的结果；非数值量则是因为 0/0 或者 Inf/Inf 型的非正常运算。需要注意的是，两个 NaN 彼此是不相等的。

除了运算造成这些异常结果外，MATLAB 也提供了专门函数可以创建这两种特别

的量，读者可以用 Inf 函数和 NaN 函数创建指定数值类型的无穷量和非数值量，默认是双精度浮点类型。

【例 1-8】 无穷量和非数值量。

解：在 MATLAB 命令行窗口输入：

```
>> x = 1/0
x =
   Inf
>> y = log(0)
y =
   - Inf
>> z = 0.0/0.0
z =
   NaN
```

1.2 向量

向量是高等数学、线性代数中讨论的概念。虽是一个数学的概念，但它同时又在力学、电磁学等许多领域中被广泛应用。电子信息学科的电磁场理论课程就以向量分析和场论作为其数学基础。

向量是一个有方向的量。在平面解析几何中，它用坐标表示成从原点出发到平面上的一点(a,b)，数据对(a,b)称为一个二维向量。立体解析几何中，则用坐标表示成(a,b,c)，数据组(a,b,c)称为三维向量。线性代数推广了这一概念，提出了 n 维向量，在线性代数中，n 维向量用 n 个元素的数据组表示。

MATLAB 讨论的向量以线性代数的向量为起点，多可达 n 维抽象空间，少可应用到解决平面和空间的向量运算问题。下面首先讨论在 MATLAB 中如何生成向量的问题。

1.2.1 向量的生成

在 MATLAB 中，生成向量主要有 3 种方案：直接输入法、冒号表达式法和函数法。现分述如下。

1. 直接输入法

在命令提示符之后直接输入一个向量，其格式是：向量名=[a1,a2,a3,…]。

【例 1-9】 直接法输入向量。

解：输入命令后运行结果如下：

```
>> A = [2,3,4,5,6],B = [1;2;3;4;5],C = [4 5 6 7 8 9]
A =
     2   3   4   5   6
B =
     1
```

```
     2
     3
     4
     5
C =
     4   5   6   7   8   9
```

2. 冒号表达式法

利用冒号表达式 a1:step:an 也能生成向量，式中 a1 为向量的第一个元素，an 为向量最后一个元素的限定值，step 是变化步长，省略步长时系统默认为 1。

【例 1-10】 用冒号表达式生成向量。

解：输入命令后运行结果如下：

```
>> A = 1:2:10;B = 1:10,C = 10: - 1:1,D = 10:2:4,E = 2: - 1:10
B =
     1   2   3   4   5   6   7   8   9   10
C =
    10   9   8   7   6   5   4   3   2   1
D =
   Empty matrix: 1 - by - 0
E =
   Empty matrix: 1 - by - 0
```

3. 函数法

有两个函数可用来直接生成向量。一个实现线性等分——linspace()；另一个实现对数等分——logspace()。

线性等分的通用格式为 A=linspace(a1,an ,n)，其中 a1 是向量的首元素，an 是向量的尾元素，n 把 a1 至 an 之间的区间分成向量的首尾之外的其他 n−2 个元素。省略 n 则默认生成 100 个元素的向量。

【例 1-11】 请在 MATLAB 命令行窗口输入以下语句，观察用线性等分函数生成向量的结果。

解：输入命令后运行结果如下：

```
>> A = linspace(1,50),B = linspace(1,30,10)
```

对数等分的通用格式为 A=logspace(a1,an,n)，其中 a1 是向量首元素的幂，即 $A(1)=10^{a1}$；an 是向量尾元素的幂，即 $A(n)=10^{an}$。n 是向量的维数。省略 n 则默认生成 50 个元素的对数等分向量。

【例 1-12】 请在 MATLAB 命令行窗口输入以下语句，观察用对数等分函数生成向量的结果。

解：输入命令后运行结果如下：

```
>> A = logspace(0,49),B = logspace(0,4,5)
```

尽管用冒号表达式和线性等分函数都能生成线性等分向量，但在使用时有以下几点区别值得注意：

(1) an 在冒号表达式中，它不一定恰好是向量的最后一个元素，只有当向量的倒数第二个元素加步长等于 an 时，an 才正好构成尾元素。如果一定要构成一个以 an 为末尾元素的向量，那么最可靠的生成方法是用线性等分函数。

(2) 在使用线性等分函数前，必须先确定生成向量的元素个数。但使用冒号表达式将依着步长和 an 的限制去生成向量，用不着去考虑元素个数的多少。

实际应用时，同时限定尾元素和步长去生成向量，有时会出现矛盾，此时必须做出取舍。要么坚持步长优先，调整尾元素限制；要么坚持尾元素限制，去修改等分步长。

1.2.2 向量的加减和数乘运算

在 MATLAB 中，维数相同的行向量之间可以相加减，维数相同的列向量也可相加减，标量数值可以与向量直接相乘除。

【例 1-13】 向量的加、减和数乘运算。

解：输入命令后运行结果如下：

```
A = [1 2 3 4 5];
B = 3:7;
C = linspace(2,4,3);
AT = A';
BT = B';
E1 = A + B,
E2 = A - B,
F = AT - BT,
G1 = 3 * A,
G2 = B/3,
H = A + C
```

其运行结果为

```
E1 =
      4     6     8    10    12
E2 =
    - 2   - 2   - 2   - 2   - 2
F =
    - 2
    - 2
    - 2
    - 2
    - 2
G1 =
```

```
    3   6   9   12   15
G2 =
    1.0000   1.3333   1.6667   2.0000   2.3333
Error using +
Matrix dimensions must agree.
```

上述实例执行后，H＝A＋C 显示了出错信息，表明维数不同的向量之间的加减法运算是非法的。

1.2.3　向量的点、叉积运算

向量的点积即数量积，叉积又称向量积或矢量积。点积、叉积甚至两者的混合积在场论中是极其基本的运算。MATLAB 是用函数实现向量点、叉积运算的。下面举例说明向量的点积、叉积和混合积运算。

1. 点积运算

点积运算（$\boldsymbol{A}\cdot\boldsymbol{B}$）的定义是参与运算的两向量各对应位置上元素相乘后，再将各乘积相加。所以向量点积的结果是标量而非向量。

点积运算函数是：dot(A,B)，A、B 是维数相同的两向量。

【例 1-14】　向量点积运算。

解：输入命令后运行结果如下：

```
A = 1:10;
B = linspace(1,10,10);
AT = A';BT = B';
e = dot(A,B),
f = dot(AT,BT)
```

其运行结果为

```
e =
   385
f =
   385
```

2. 叉积运算

在数学描述中，向量 $\boldsymbol{A}$、$\boldsymbol{B}$ 的叉积是一新向量 $\boldsymbol{C}$，$\boldsymbol{C}$ 的方向垂直于 $\boldsymbol{A}$ 与 $\boldsymbol{B}$ 所决定的平面。用三维坐标表示时

$$\boldsymbol{A} = A_x\boldsymbol{i} + A_y\boldsymbol{j} + A_z\boldsymbol{k}$$

$$\boldsymbol{B} = B_x\boldsymbol{i} + B_y\boldsymbol{j} + B_z\boldsymbol{k}$$

$$\boldsymbol{C} = \boldsymbol{A}\times\boldsymbol{B} = (A_yB_z - A_zB_y)\boldsymbol{i} + (A_zB_x - A_xB_z)\boldsymbol{j} + (A_xB_y - A_yB_x)\boldsymbol{k}$$

叉积运算的函数是：cross(A,B)，该函数计算的是 A、B 叉积后各分量的元素值，且

A、B 只能是三维向量。

【例 1-15】 合法向量叉积运算。

解：输入命令后运行结果如下：

```
A=1:3,
B=3:5
E=cross(A,B)
```

其运行结果为

```
A =
1   2   3
B =
3   4   5
E =
-2   4   -2
```

【例 1-16】 非法向量叉积运算(不等于三维的向量做叉积运算)。

解：输入命令后运行结果如下：

```
A=1:4,
B=3:6,
C=[1 2],
D=[3 4]
E=cross(A,B),
F=cross(C,D)
```

其运行结果为

```
A =
1   2   3   4
B =
3   4   5   6
C =
1   2
D =
4
Error using ==> cross
A and B must have at least one dimension of length 3.
```

3. 混合积运算

综合运用上述两个函数就可实现点积和叉积的混合运算，该运算也只能发生在三维向量之间，现示例如下。

【例 1-17】 向量混合积示例。

解：输入命令后运行结果如下：

```
A=[1 2 3],
B=[3 3 4],
C=[3 2 1]
D=dot(C,cross(A,B))
```

其运行结果为

```
>>A =
1   2   3
>>B =
3   3   4
>>C =
3   2   1
>>D =
4
```

1.3 数组

数组运算是 MATLAB 计算的基础。由于 MATLAB 面向对象的特性，这种数值数组成为 MATLAB 最重要的一种内建数据类型，而数组运算就是定义这种数据结构的方法。本节将系统地列出具备数组运算能力的函数名称，为兼顾一般性，以二维数组的运算为例，读者可推广至多维数组和多维矩阵的运算。

下面将介绍在 MATLAB 中如何建立数组，以及数组的常用操作等，包括数组的算术运算、关系运算和逻辑运算。

1.3.1 数组的创建和操作

在 MATLAB 中一般使用方括号“[ ]”、逗号“,”、空格号和分号“;”来创建数组。数组中同一行的元素使用逗号或空格进行分隔，不同行之间用分号进行分隔。

【例 1-18】 创建空数组、行向量、列向量示例。

在命令行窗口中输入如下语句：

```
clear all
A=[]
B=[6 5 4 3 2 1]
C=[6,5,4,3,2,1]
D=[6;5;4;3;2;1]
E=B'              %转置
```

命令行窗口中的输出结果如下：

```
A =   []
B =   6   5   4   3   2   1
C =   6   5   4   3   2   1
D =
     6
     5
     4
     3
     2
     1
E =
     6
     5
     4
     3
     2
     1
```

【例 1-19】 访问数组示例。

在命令行窗口中输入如下语句：

```
clear all
clc
A = [6 5 4 3 2 1]
a1 = A(1)                    % 访问数组第一个元素
a2 = A(1:3)                  % 访问数组第 1、2、3 个元素
a3 = A(3:end)                % 访问数组第 3 个到最后一个元素
a4 = A(end: - 1:1)           % 数组元素反序输出
a5 = A([1 6])                % 访问数组第 1 个及第 6 个元素
```

命令行窗口中的输出结果如下：

```
A =    6   5   4   3   2   1
a1  =   6
a2  =   6   5   4
a3  =   4   3   2   1
a4  =   1   2   3   4   5   6
a5  =   6   1
```

【例 1-20】 子数组的赋值(assign)示例。

在命令行窗口中输入如下语句：

```
clear all
clc
A = [6 5 4 3 2 1]
A(3)  = 0
A([1 4]) = [1 1]
```

命令行窗口中的输出结果如下：

```
A =   6   5   4   3   2   1
A =   6   5   0   3   2   1
A =   1   5   0   1   2   1
```

在 MATLAB 中还可以通过其他各种方式创建数组，具体如下。

1. 通过冒号创建一维数组

在 MATLAB 中，通过冒号创建一维数组的代码如下：

```
X = A:step:B
```

其中，A 是创建一维数组的第一个变量，step 是每次递增或递减的数值，直到最后一个元素和 B 的差的绝对值小于等于 step 的绝对值为止。

【例 1-21】 通过冒号创建一维数组示例。

在命令行窗口中输入如下语句：

```
clear all
clc
A = 2:6
B = 2.1:1.5:6
C = 2.1: - 1.5: - 6
D = 2.1: - 1.5:6
```

命令行窗口中的输出结果如下：

```
A =
    2   3   4   5   6
B =
    2.1000    3.6000    5.1000
C =
    2.1000    0.6000   - 0.9000   - 2.4000   - 3.9000   - 5.4000
D =
    空的 1×0 double 行矢量
```

2. 通过 logspace 函数创建一维数组

MATLAB 常用 logspace()函数创建一维数组，该函数的调用方式如下。

y= logspace(a,b)：该函数创建行向量 y，第一个元素为 10^a，最后一个元素为 10^b，形成总数为 50 个元素的等比数列。

y = logspace(a,b,n)：该函数创建行向量 y，第一个元素为 10^a，最后一个元素为 10^b，形成总数为 n 个元素的等比数列。

【例 1-22】 通过 logspace 函数创建一维数组示例。

在命令行窗口中输入如下语句：

```
clear all
clc
format short;
A = logspace(1,2,20)
B = logspace(1,2,10)
```

命令行窗口中的输出结果如下：

```
A =
  1 至 14 列
  10.0000  11.2884  12.7427  14.3845  16.2378  18.3298  20.6914  23.3572  26.3665
29.7635  33.5982  37.9269  42.8133  48.3293

  15 至 20 列
  54.5559  61.5848  69.5193  78.4760  88.5867  100.0000

B =
  10.0000  12.9155  16.6810  21.5443  27.8256  35.9381  46.4159  59.9484  77.4264
100.0000
```

3. 通过 linspace 函数创建一维数组

MATLAB 常用 linspace()函数创建一维数组，该函数的调用方式如下。

y= linspace (a,b)：该函数创建行向量 y，第一个元素为 a，最后一个元素为 b，形成总数为 100 个元素的等比数列。

y =linspace (a,b,n)：该函数创建行向量 y，第一个元素为 a ，最后一个元素为 b，形成总数为 n 个元素的等比数列。

【例 1-23】 通过 linspace 函数创建一维数组示例。

在命令行窗口中输入如下语句：

```
clear all
clc
format short;
A = linspace(1,100)
B = linspace(1,36,12)
C= linspace(1,36,1)
```

命令行窗口中的输出结果如下：

```
A =
  1 至 23 列
     1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16   17   18   19   20   21   22
  23

  24 至 46 列
```

```
    24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43
  44  45  46

  47 至 69 列
    47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66
  67  68  69

  70 至 92 列
    70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92

  93 至 100 列
    93  94  95  96  97  98  99  100

B =
    1.0000  4.1818  7.3636  10.5455  13.7273  16.9091  20.0909  23.2727  26.4545
29.6364  32.8182  36.0000
C =
    36
```

1.3.2 数组的常见运算

1. 数组的算术运算

数组的运算是从数组的单个元素出发，针对每个元素进行的运算。在 MATLAB 中，一维数组的基本运算包括加、减、乘、左除、右除和乘方。

数组的加减运算：通过格式 **A**+**B** 或 **A**-**B** 可实现数组的加减运算。但是运算规则要求数组 **A** 和 **B** 的维数相同。

提示：如果两个数组的维数不相同，则将给出错误的信息。

【例 1-24】 数组的加减运算示例。

在命令行窗口中输入如下语句：

```
clear all
clc
A=[1 5 6 8 9 6]
B=[9 85 6 2 4 0]
C=[1 1 1 1 1]
D=A+B               %加法
E=A-B               %减法
F=A*2
G=A+3               %数组与常数的加法
H=A-C
```

命令行窗口中的输出结果如下：

```
A =
     1     5     6     8     9     6
B =
     9    85     6     2     4     0
C =
     1     1     1     1     1
D =
    10    90    12    10    13     6
E =
    -8   -80     0     6     5     6
F =
     2    10    12    16    18    12
G =
     4     8     9    11    12     9
```

矩阵维度必须一致。

数组的乘除运算：通过格式“. *”或“. /”可实现数组的乘除运算。但是运算规则要求数组 ***A*** 和 ***B*** 的维数相同。

乘法：数组 ***A*** 和 ***B*** 的维数相同，运算为数组对应元素相乘，计算结果与 ***A*** 和 ***B*** 是相同维数的数组。

除法：数组 ***A*** 和 ***B*** 的维数相同，运算为数组对应元素相除，计算结果与 ***A*** 和 ***B*** 是相同维数的数组。

右除和左除的关系：***A***. /***B***=***B***.***A***，其中 ***A*** 是被除数，***B*** 是除数。

提示：如果两个数组的维数不相同，则将给出错误的信息。

【例 1-25】 数组的乘法示例。

解：在命令行窗口中输入如下语句：

```
clear all
clc
A=[1 5 6 8 9 6]
B=[9 5 6 2 4 0]
C=A.* B              %数组的点乘
D=A * 3              %数组与常数的乘法
```

命令行窗口中的输出结果如下：

```
A =
     1     5     6     8     9     6
B =
     9     5     6     2     4     0
C =
     9    25    36    16    36     0
D =
     3    15    18    24    27    18
```

【例 1-26】 数组的除法示例。

解：在命令行窗口中输入如下语句：

```
clear all
clc
A=[1 5 6 8 9 6]
B=[9 5 6 2 4 0]
C=A.\B            %数组和数组的左除
D=A./B            %数组和数组的右除
E=A./3            %数组与常数的除法
F=A/3
```

命令行窗口中的输出结果如下：

```
A =
     1   5   6   8   9   6
B =
     9   5   6   2   4   0
C =
    9.0000   1.0000   1.0000   0.2500   0.4444   0
D =
    0.1111   1.0000   1.0000   4.0000   2.2500   Inf
E =
    0.3333   1.6667   2.0000   2.6667   3.0000   2.0000
F =
    0.3333   1.6667   2.0000   2.6667   3.0000   2.0000
```

通过乘方格式“.^”实现数组的乘方运算。数组的乘方运算包括数组间的乘方运算、数组与某个具体数值的乘方运算以及常数与数组的乘方运算。

【例 1-27】 数组的乘方示例。

解：在命令行窗口中输入如下语句：

```
clear all
clc
A=[1 5 6 8 9 6]
B=[9 5 6 2 4 0]
C=A.^B            %数组的乘方
D=A.^3            %数组与某个具体数值的乘方
E=3.^A            %常数与数组的乘方
```

命令行窗口中的输出结果如下：

```
A =
    1   5   6   8   9   6
B =
    9   5   6   2   4   0
C =
    1   3125   46656   64   6561   1
```

```
D =
     1   125   216   512   729   216
E =
     3   243   729   6561   19683   729
```

通过函数 dot()可实现数组的点积运算，但是运算规则要求数组 **A** 和 **B** 的维数相同，其调用格式如下：

```
C = dot(A,B)
C = dot(A,B,dim)
```

【例 1-28】 数组的点积示例。

解：在命令行窗口中输入如下语句：

```
clear all
clc
A = [1 5 6 8 9 6]
B = [9 5 6 2 4 0]
C = dot(A,B)                % 数组的点积
D = sum(A. * B)             % 数组元素的乘积之和
```

命令行窗口中的输出结果如下：

```
A =
     1   5   6   8   9   6
B =
     9   5   6   2   4   0
C =
     122
D =
     122
```

2. 数组的关系运算

在 MATLAB 中提供了 6 种数组关系运算符，即<(小于)、<=(小于等于)、>(大于)、>=(大于等于)、==(恒等于)、~=(不等于)。

关系运算的运算法则如下：

当两个比较量是标量时，直接比较两个数的大小。若关系成立，则返回的结果为 1，否则为 0。

当两个比较量是维数相等的数组时，逐一比较两个数组相同位置的元素，并给出比较结果。最终的关系运算结果是一个与参与比较的数组维数相同的数组，其组成元素为 0 或 1。

【例 1-29】 数组的关系运算示例。

解：在命令行窗口中输入如下语句：

```
clear all
clc
A=[1 5 6 8 9 6]
B=[9 5 6 2 4 0]
C=A<6                      %数组与常数比较,小于
D=A>=6                     %数组与常数比较,大于等于
E=A<B                      %数组与数组比较,小于
F=A==B                     %数组与数组比较,恒等于
```

命令行窗口中的输出结果如下：

```
A =
    1   5   6   8   9   6
B =
    9   5   6   2   4   0
C =
    1×6 logical 数组
    1   1   0   0   0   0
D =
    1×6 logical 数组
    0   0   1   1   1   1
E =
    1×6 logical 数组
    1   0   0   0   0   0
F =
    1×6 logical 数组
    0   1   1   0   0   0
```

3. 数组的逻辑运算

在 MATLAB 中数组提供了 3 种数组逻辑运算符，即 &(与)、|(或)和～(非)。逻辑运算的运算法则如下：

如果是非零元素则为真，用 1 表示；反之是零元素则为假，用 0 表示。

当两个比较量是维数相等的数组时，逐一比较两个数组相同位置的元素，并给出比较结果。最终的关系运算结果是一个与参与比较的数组维数相同的数组，其组成元素为 0 或 1。

与运算($a\&b$)时，a、b 全为非零，则为真，运算结果为 1；或运算($a|b$)时，只要 a、b 有一个为非零，则运算结果为 1；非运算($\sim a$)时，若 a 为 0，运算结果为 1，a 为非零，运算结果为 0。

【例 1-30】 数组的逻辑运算示例。

在命令行窗口中输入如下语句：

```
clear all
clc
A=[1 5 6 8 9 6]
B=[9 5 6 2 4 0]
```

```
C = A&B          % 与
D = A|B          % 或
E = ~B           % 非
```

命令行窗口中的输出结果如下：

```
A =
    1  5  6  8  9  6
B =
    9  5  6  2  4  0
C =
    1×6 logical 数组
    1  1  1  1  1  0
D =
    1×6 logical 数组
    1  1  1  1  1  1
E =
    1×6 logical 数组
    0  0  0  0  0  1
```

1.4 矩阵

MATLAB 简称矩阵实验室，对于矩阵的运算，MATLAB 软件有着得天独厚的优势。

生成矩阵的方法有直接输入矩阵元素；对已知矩阵进行矩阵组合、矩阵转向、矩阵移位操作；读取数据文件；使用函数直接生成特殊矩阵。表 1-13 列出了常用的特殊矩阵生成函数。

表 1-13 常用的特殊矩阵生成函数

函数名	说　明	函数名	说　明
zeros	全 0 矩阵	eye	单位矩阵
ones	全 1 矩阵	company	伴随矩阵
rand	均匀分布随机矩阵	hilb	Hilbert 矩阵
randn	正态分布随机分布	invhilb	Hilbert 逆矩阵
magic	魔方矩阵	vander	Vander 矩阵
diag	对角矩阵	pascal	Pascal 矩阵
triu	上三角矩阵	hadamard	Hadamard 矩阵
tril	下三角矩阵	hankel()	Hankel 矩阵

1.4.1 矩阵生成

【例 1-31】 随机矩阵输入。

解：在 MATLAB 输入以下代码：

```
A = rand(5)
```

运行得到如下结果：

```
A =
     0.0512  0.4141  0.0594  0.0557  0.5681
     0.8698  0.1400  0.3752  0.6590  0.0432
     0.0422  0.2867  0.8687  0.9065  0.4148
     0.0897  0.0919  0.5760  0.1293  0.3793
     0.0541  0.1763  0.8402  0.7751  0.7090
```

A 中第一列如下：

```
>> A(:,1)
ans =
     0.7577
     0.7431
     0.3922
     0.6555
     0.1712
```

A 中第二列如下：

```
>> A(:,2)
ans =
     0.7060
     0.0318
     0.2769
     0.0462
     0.0971
```

A 中第三、四、五列如下：

```
>> A(:,3:5)
ans =
     0.8235  0.4387  0.4898
     0.6948  0.3816  0.4456
     0.3171  0.7655  0.6463
     0.9502  0.7952  0.7094
     0.0344  0.1869  0.7547
```

A 中第一行如下：

```
>> A(1,:)
ans =
     0.7577  0.7060  0.8235  0.4387  0.4898
```

A 中第二行如下：

```
>> A(2,:)
ans =
    0.7431  0.0318  0.6948  0.3816  0.4456
```

A 中第三、四、五行如下：

```
>> A(3:5,:)
ans =
    0.3922  0.2769  0.3171  0.7655  0.6463
    0.6555  0.0462  0.9502  0.7952  0.7094
    0.1712  0.0971  0.0344  0.1869  0.7547
```

【例 1-32】 矩阵的乘法运算。

解：在 MATLAB 输入以下代码：

```
>> A^2
```

运行如下：

```
ans =
    0.4011  0.2015  0.7194  0.7772  0.4955
    0.2436  0.5555  0.8460  0.5994  0.9364
    0.3919  0.4631  1.7354  1.4175  1.0347
    0.1410  0.2939  0.9334  0.8985  0.6118
    0.2995  0.4842  1.8414  1.5305  1.1836
```

【例 1-33】 矩阵的点乘运算。

解：在 MATLAB 输入以下代码：

```
A.^2
```

运行如下：

```
ans =
    0.0026  0.1715  0.0035  0.0031  0.3227
    0.7565  0.0196  0.1408  0.4343  0.0019
    0.0018  0.0822  0.7547  0.8217  0.1721
    0.0080  0.0085  0.3318  0.0167  0.1439
    0.0029  0.0311  0.7059  0.6008  0.5026
```

【例 1-34】 矩阵的除法运算。

解：在 MATLAB 输入以下代码：

```
>> A^2\A.^2
```

运行如下：

```
ans =
     0.2088    0.5308   -0.4762    0.8505   -0.0382
     1.3631   -0.1769    1.1661    0.8143   -4.2741
    -0.3247   -0.0898    1.5800    2.7892   -1.0326
    -0.5223    0.0537   -0.5715   -2.4802    0.4729
     0.5725    0.0345   -1.4792   -1.1727    3.1778
```

【例 1-35】 矩阵的减法运算。

解：在 MATLAB 输入以下代码：

```
>> A^2-A.^2
```

运行如下：

```
ans =
     0.3984  0.0300  0.7159  0.7741  0.1728
    -0.5129  0.5359  0.7052  0.1652  0.9345
     0.3901  0.3810  0.9807  0.5958  0.8626
     0.1330  0.2854  0.6016  0.8818  0.4679
     0.2965  0.4531  1.1355  0.9297  0.6809
```

【例 1-36】 矩阵的加法运算。

解：在 MATLAB 输入以下代码：

```
>> A^2+A.^2
```

运行如下：

```
ans =
    0.4037  0.3730  0.7229  0.7803  0.8182
    1.0001  0.5751  0.9868  1.0337  0.9383
    0.3937  0.5453  2.4901  2.2392  1.2068
    0.1491  0.3023  1.2652  0.9152  0.7558
    0.3024  0.5153  2.5473  2.1314  1.6862
```

【例 1-37】 Hankel 矩阵求解。

解：在 MATLAB 输入以下代码：

```
%% hankel 矩阵
clc,clear,close all
c=[1:3],
r=[3:9],
H=hankel(c,r)
```

运行程序输出结果如下：

```
c =
     1   2   3
r =
     3   4   5   6   7   8   9
H =
     1   2   3   4   5   6   7
     2   3   4   5   6   7   8
     3   4   5   6   7   8   9
```

【例 1-38】 Hilbert 矩阵及 Hilbert 逆矩阵生成。

解：在 MATLAB 输入以下代码：

```
clear all
clc,
A = hilb(5)
```

运行程序输出结果如下：

```
A =
    1.0000   0.5000   0.3333   0.2500   0.2000
    0.5000   0.3333   0.2500   0.2000   0.1667
    0.3333   0.2500   0.2000   0.1667   0.1429
    0.2500   0.2000   0.1667   0.1429   0.1250
    0.2000   0.1667   0.1429   0.1250   0.1111
```

更改输出格式如下：

```
format rat
A
```

运行程序输出结果如下：

```
A =
       1            1/2          1/3          1/4          1/5
       1/2          1/3          1/4          1/5          1/6
       1/3          1/4          1/5          1/6          1/7
       1/4          1/5          1/6          1/7          1/8
       1/5          1/6          1/7          1/8          1/9
```

【例 1-39】 Hilbert 逆矩阵求解。

解：在 MATLAB 输入以下代码：

```
A = invhilb(5)
```

运行程序输出结果如下：

```
A =
       25       -300       1050      -1400        630
     -300       4800     -18900      26880     -12600
     1050     -18900      79380    -117600      56700
    -1400      26880    -117600     179200     -88200
      630     -12600      56700     -88200      44100
```

1.4.2 向量的生成

向量是指单行或单列的矩阵，是组成矩阵的基本元素之一。在求某些函数值或曲线时，常常要设定自变量的一系列值，因此除了直接使用“[]”生成向量，MATLAB还提供了两种为等间隔向量赋值的简单方法。

1. 使用冒号表达式生成向量

冒号表达式的格式为 x=[初值 x_0:增量:终值 x_n]。这里需要注意以下几点：

(1) 生成的向量尾元素并不一定是终值 x_n，当 x_n-x_0 恰好为增量的整数倍时，x_n 才为尾元素。

(2) 当 $x_n>x_0$ 时，增量必须为正值；当 $x_n<x_0$ 时，增量必须为负值；当 $x_n=x_0$ 时，向量只有一个元素。

(3) 当增量为1时，增量值可以略去，直接写成 x=[初值 x_0:终值 x_n]。

(4) 方括号“[]”可以删去。

2. 使用linspace函数生成向量

linspace函数的调用格式为 x=linspace(初值 x_1,终值 x_n,点数 n)，点数 n 也可不写，此时默认 n=100。

【例1-40】 等间隔向量赋值。

解：在MATLAB输入以下代码：

```
>> t = 1:3:20
```

运行程序输出结果如下：

```
t =
     1    4    7   10   13   16   19
```

命令如下：

```
>> t = 10:-3:-20
```

运行程序输出结果如下：

```
t =
  Columns 1 through 9
     10    7    4    1   -2   -5   -8   -11   -14
  Columns 10 through 11
   -17   -20
```

命令如下：

```
>> t = 1:2:1
```

运行程序输出结果如下：

```
t =
    1
```

命令如下：

```
>> t = 1:5
```

运行程序输出结果如下：

```
t =
     1   2   3   4   5
```

命令如下：

```
>> t = linspace(1,10,5)
```

运行程序输出结果如下：

```
t =
    1.0000   3.2500   5.5000   7.7500   10.0000
```

【例 1-41】 有时需要生成对数等比向量，此时可用 logspace 函数，其调用格式为 x=logspace(初值 x_1，终值 x_n，点数 n)，它表示从 10 的 x_1 次幂到 x_n 次幂等比生成 n 个点。

解：在 MATLAB 输入以下代码：

```
>> t = logspace(0,1,15)
```

运行程序输出结果如下：

```
t =
  Columns 1 through 5
    1.0000   1.1788   1.3895   1.6379   1.9307
  Columns 6 through 10
```

```
    2.2758   2.6827   3.1623   3.7276   4.3940
  Columns 11 through 15
    5.1795   6.1054   7.1969   8.4834   10.0000
```

矩阵的加、减、乘、除、比较运算和逻辑运算等代数运算是 MATLAB 数值计算最基础的部分。本节将重点介绍这些运算。

1.4.3 矩阵加减运算

进行矩阵加法、减法运算的前提是参与运算的两个矩阵或多个矩阵必须具有相同的行数和列数，即 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 等多个矩阵均为 $m\times n$ 矩阵；或者其中有一个或多个矩阵为标量。

在上述前提下，对于同型的两个矩阵，其加减法定义如下：

$\boldsymbol{C}=\boldsymbol{A}\pm\boldsymbol{B}$，矩阵 $\boldsymbol{C}$ 的各元素 $\boldsymbol{C}_{mn}=\boldsymbol{A}_{mn}+\boldsymbol{B}_{mn}$。

当其中含有标量 x 时，$\boldsymbol{C}=\boldsymbol{A}\pm x$，矩阵 $\boldsymbol{C}$ 的各元素 $\boldsymbol{C}_{mn}=\boldsymbol{A}_{mn}+x$。

由于矩阵的加法运算归结为其元素的加法运算，容易验证，因此矩阵的加法运算满足下列运算律：

（1）交换律：$\boldsymbol{A}+\boldsymbol{B}=\boldsymbol{B}+\boldsymbol{A}$。

（2）结合律：$\boldsymbol{A}+(\boldsymbol{B}+\boldsymbol{C})=(\boldsymbol{A}+\boldsymbol{B})+\boldsymbol{C}$。

（3）存在零元：$\boldsymbol{A}+\boldsymbol{0}=\boldsymbol{0}+\boldsymbol{A}=\boldsymbol{A}$。

（4）存在负元：$\boldsymbol{A}+(-\boldsymbol{A})=(-\boldsymbol{A})+\boldsymbol{A}$。

【例 1-42】 矩阵加减法运算示例。

已知矩阵 $\boldsymbol{A}=$ [10 5 79 4 2;1 0 66 8 2;4 6 1 1 1]，矩阵 $\boldsymbol{B}=$ [9 5 3 4 2;1 0 4 −23 2;4 6 −1 1 0]，行向量 $\boldsymbol{C}=$ [2 1]，标量 $x=20$，试求 $\boldsymbol{A}+\boldsymbol{B}$、$\boldsymbol{A}-\boldsymbol{B}$、$\boldsymbol{A}+\boldsymbol{B}+x$、$\boldsymbol{A}-x$、$\boldsymbol{A}-\boldsymbol{C}$。

解：在 MATLAB 输入以下代码：

```
clear all
clc
A = [10 5 79 4 2;1 0 66 8 2;4 6 1 1 1];
B = [9 5 3 4 2;1 0 4 -23 2;4 6 -1 1 0];
x = 20;
C = [2 1];
ApB =  A + B
AmB =  A - B
ApBpX =  A + B + x
AmX =  A - x
AmC =  A - C
```

得到的结果为

```
ApB =
    19   10   82     8   4
     2    0   70   -15   4
```

```
    8   12    0      2   1
AmB =
    1   0   76    0   0
    0   0   62   31   0
    0   0    2    0   1
ApBpX =
   39   30   102   28   24
   22   20    90    5   24
   28   32    20   22   21
AmX =
  -10   -15    59   -16   -18
  -19   -20    46   -12   -18
  -16   -14   -19   -19   -19
错误使用 -
矩阵维度必须一致.
```

在 $\boldsymbol{A}-\boldsymbol{C}$ 的运算中，MATLAB 返回错误信息，并提示矩阵的维数必须相等。这也证明了矩阵进行加减法运算必须满足一定的前提条件。

1.4.4 矩阵乘法运算

MATLAB 中矩阵的乘法运算包括两种：数与矩阵的乘法；矩阵与矩阵的乘法。

1. 数与矩阵的乘法

由于单个数在 MATLAB 中是以标量来存储的，因此数与矩阵的乘法也可以称为标量与矩阵的乘法。

设 x 为一个数，$\boldsymbol{A}$ 为矩阵，则定义 x 与 $\boldsymbol{A}$ 的乘积 $\boldsymbol{C}=x\boldsymbol{A}$ 仍为一个矩阵，$\boldsymbol{C}$ 的元素就是用数 x 乘矩阵 $\boldsymbol{A}$ 中对应的元素而得到，即 $\boldsymbol{C}_{mn}=x\boldsymbol{A}_{mn}$。数与矩阵的乘法满足下列运算律：

$$1\boldsymbol{A}=\boldsymbol{A}$$
$$x(\boldsymbol{A}+\boldsymbol{B})=x\boldsymbol{A}+x\boldsymbol{B}$$
$$(x+y)\boldsymbol{A}=x\boldsymbol{A}+y\boldsymbol{A}$$
$$(xy)\boldsymbol{A}=x(y\boldsymbol{A})=y(x\boldsymbol{A})$$

【例 1-43】 矩阵数乘示例。

已知矩阵 $\boldsymbol{A}=[0\ 3\ 3;1\ 1\ 0;-1\ 2\ 3]$，$\boldsymbol{E}$ 是 3 阶单位矩阵，$\boldsymbol{E}=[1\ 0\ 0;0\ 1\ 0;0\ 0\ 1]$，试求表达式 $2\boldsymbol{A}+3\boldsymbol{E}$。

解：在 MATLAB 输入以下代码：

```
A = [0 3 3;1 1 0;-1 2 3];
E = eye(3);
R=2*A+3*E
```

得到的结果为

```
R =
     3   6   6
     2   5   0
    -2   4   9
```

2. 矩阵与矩阵的乘法

两个矩阵的乘法必须满足被乘矩阵的列数与乘矩阵的行数相等。设矩阵 $\boldsymbol{A}$ 为 $m\times h$ 矩阵，$\boldsymbol{B}$ 为 $h\times n$ 矩阵，则两矩阵的乘积 $\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$ 为一个矩阵，且 $\boldsymbol{C}_{mn}=\sum_{h=1}^{H}\boldsymbol{A}_{mh}\times\boldsymbol{B}_{hn}$ 。

矩阵之间的乘法不遵循交换律，即 $\boldsymbol{A}\times\boldsymbol{B}\neq\boldsymbol{B}\times\boldsymbol{A}$。但矩阵乘法遵循下列运算律：

结合律：$(\boldsymbol{A}\times\boldsymbol{B})\times\boldsymbol{C}=\boldsymbol{A}\times(\boldsymbol{B}\times\boldsymbol{C})$。

左分配律：$\boldsymbol{A}\times(\boldsymbol{B}+\boldsymbol{C})=\boldsymbol{A}\times\boldsymbol{B}+\boldsymbol{A}\times\boldsymbol{C}$。

右分配律：$(\boldsymbol{B}+\boldsymbol{C})\times\boldsymbol{A}=\boldsymbol{B}\times\boldsymbol{A}+\boldsymbol{C}\times\boldsymbol{A}$。

单位矩阵的存在性：$\boldsymbol{E}\times\boldsymbol{A}=\boldsymbol{A}$，$\boldsymbol{A}\times\boldsymbol{E}=\boldsymbol{A}$。

【例 1-44】 矩阵乘法的示例。

已知矩阵 $\boldsymbol{A}=[2\ 1\ 4\ 0;1\ -1\ 3\ 4]$，矩阵 $\boldsymbol{B}=[1\ 3\ 1;0\ -1\ 2;1\ -3\ 1;4\ 0\ -2]$，试求矩阵乘积 $\boldsymbol{AB}$ 及 $\boldsymbol{BA}$。

解：在 MATLAB 输入以下代码：

```
A = [2 1 4 0;1 -1 3 4];
B = [1 3 1;0 -1 2;1 -3 1;4 0 -2];
R1 = A*B
R2 = B*A
```

程序运行结果如下：

```
R1 =
     6    -7     8
    20    -5    -6
错误使用 *
内部矩阵维度必须一致.      % 由于不满足矩阵的乘法条件，故 BA 无法计算
```

1.4.5 矩阵的除法运算

矩阵的除法是乘法的逆运算，分为左除和右除两种，分别用运算符号“\”和“/”表示。如果矩阵 $\boldsymbol{A}$ 和矩阵 $\boldsymbol{B}$ 是标量，那么 $\boldsymbol{A}/\boldsymbol{B}$ 和 $\boldsymbol{A}\backslash\boldsymbol{B}$ 是等价的。对于一般的二维矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，当进行 $\boldsymbol{A}\backslash\boldsymbol{B}$ 运算时，要求 $\boldsymbol{A}$ 的行数与 $\boldsymbol{B}$ 的行数相等；当进行 $\boldsymbol{A}/\boldsymbol{B}$ 运算时，要求 $\boldsymbol{A}$ 的列数与 $\boldsymbol{B}$ 的列数相等。

【例 1-45】 矩阵除法的示例。

设矩阵 $\boldsymbol{A}=[1\ 2;1\ 3]$，矩阵 $\boldsymbol{B}=[1\ 0;1\ 2]$，试求 $\boldsymbol{A}\backslash\boldsymbol{B}$ 和 $\boldsymbol{A}/\boldsymbol{B}$。

解：在 MATLAB 输入以下代码：

```
A = [1 2;1 3];
B = [1 0;1 2];
R1 = A\B
R2 = A/B
```

程序运行结果为

```
R1 =
     1   -4
     0    2
R2 =
                     0   1.000000000000000
  -0.500000000000000   1.500000000000000
```

1.4.6 矩阵的分解运算

矩阵的分解常用于求解线性方程组，常用的矩阵的分解运算如表 1-14 所示。

表 1-14 MATLAB 矩阵分解函数

函数名	说 明	函数名	说 明
eig	特征值分解	chol	Cholesky 分解
svd	奇异值分解	qr	QR 分解
lu	LU 分解	schur	Schur 分解

【例 1-46】 矩阵分解运算。

解：在 MATLAB 输入以下代码：

```
>> A = [8,1,6;3,5,7;4,9,2];
>> [U,S,V] = svd(A)          % 矩阵的奇异值分解,A = U * S * V'
```

运行程序输出结果如下：

```
U =
    -0.5774    0.7071    0.4082
    -0.5774    0.0000   -0.8165
    -0.5774   -0.7071    0.4082
S =
   15.0000         0         0
         0    6.9282         0
         0         0    3.4641
V =
    -0.5774    0.4082    0.7071
    -0.5774   -0.8165   -0.0000
    -0.5774    0.4082   -0.7071
```

1.5 字符串

MATLAB 虽有字符串概念，但和 C 语言一样，仍将其视为一个一维字符数组对待。因此本节针对字符串的运算或操作，同样对字符数组也有效。

1.5.1 字符串变量与一维字符数组

当把某个字符串赋值给一个变量后，这个变量便因取得这一字符串而被 MATLAB 作为字符串变量来识别。

当观察 MATLAB 的工作区窗口时，字符串变量的类型是字符数组类型（即 char array）。而从工作区窗口去观察一个一维字符数组时，也发现它具有与字符串变量相同的数据类型。由此推知，字符串与一维字符数组在运算处理和操作过程中是等价的。

1. 给字符串变量赋值

用一个赋值语句即可完成字符串变量的赋值操作，现举例如下。

【例 1-47】 将 3 个字符串分别赋值给 S1、S2、S3 这 3 个变量。

解：输入命令后运行结果如下：

```
>> S1 = 'go home',S2 = '朝闻道,夕死可矣',S3 = 'go home. 朝闻道,夕死可矣'
S1 =
go home
S2 =
朝闻道,夕死可矣
S3 =
go home. 朝闻道,夕死可矣
```

2. 一维字符数组的生成

因为向量的生成方法就是一维数组的生成方法，而一维字符数组也是数组，与数值数组的不同是字符数组中的元素是一个个字符而非数值。因此，原则上生成向量的方法就能生成字符数组。当然最常用的还是直接输入法。

【例 1-48】 用 3 种方法生成字符数组。

解：输入命令后运行结果如下：

```
>> Sa = ['I love my teacher,  ' 'I' ' love truths '  'more profoundly.']
Sa =
I love my teacher,  I love truths more profoundly.
>> Sb = char('a':2:'r')
Sb =
acegikmoq
>> Sc = char(linspace('e','t',10))
```

```
Sc =
efhjkmoprt
```

在例 1-48 中，char()是一个将数值转换成字符串的函数。另外，请注意观察 Sa 在工作区窗口中的各项数据，尤其是 size 的大小，不要以为它只有 4 个元素，从中体会 Sa 作为一个字符数组的真正含义。

1.5.2 对字符串的多项操作

对字符串的操作主要由一组函数实现，这些函数中有求字符串长度和矩阵阶数的 length()和 size()，有进行字符串和数值相互转换的 double()和 char()等。下面举例说明用法。

1. 求字符串长度

length()和 size()虽然都能检测字符串、数组以及矩阵的大小，但在用法上有区别。length()只能从它们各维中挑出最大维的数值大小，而 size()则以一个向量的形式给出所有各维的数值大小。两者的关系是 length()＝max(size())。请仔细体会下面的举例。

【例 1-49】 length()和 size()函数的用法。

解：输入命令后运行结果如下：

```
>> Sa = ['I love my teacher, ' 'I' ' love truths ' 'more profoundly.'];
>> length(Sa)
ans =
       50
>> size(Sa)
ans =
       50
```

2. 字符串与一维数值数组的相互转换

字符串是由若干字符组成的。在 ASCII 码中，每个字符对应一个数值编码，例如字符 A 对应 65。如此一来，字符串又可在一个一维数值数组之间找到某种对应关系。这就构成了字符串与数值数组之间相互转换的基础。

【例 1-50】 用 abs()、double()、char()和 setstr()实现字符串与数值数组的相互转换。

解：输入命令后运行结果如下：

```
>> S1 = 'I am nobody';
>> As1 = abs(S1)
As1 =
       73   32   97   109   32   110   111   98   111   100   121
```

```
>> As2 = double(S1)
As2 =
      73   32   97   109   32   110   111   98   111   100   121
>> char(As2)
ans =
I am nobody
>> setstr(As2)
ans =
I am nobody
```

3. 比较字符串

strcmp(S1,S2)是 MATLAB 的字符串比较函数。当 S1 与 S2 完全相同时，返回值为 1；否则，返回值为 0。

【例 1-51】 strcmp()的用法。

解：输入命令后运行结果如下：

```
>> S1 = 'I am nobody';
>> S2 = 'I am nobody.';
>> strcmp(S1,S2)
ans = 0

>> strcmp(S1,S1) ans =
1
```

4. 查找字符串

findstr(S,s)是从某个长字符串 S 中查找子字符串 s 的函数。返回的结果值是子串在长串中的起始位置。

【例 1-52】 findstr()的用法。

解：输入命令后运行结果如下：

```
>> S = 'I believe that love is the greatest thing in the world.';
>> findstr(S,'love')
ans = 16
```

5. 显示字符串

disp()是一个原样输出其中内容的函数，它经常在程序中作提示说明用。其用法见下例。

【例 1-53】 disp()的用法。

解：输入命令后运行结果如下：

```
>> disp('两串比较的结果是：'),Result = strcmp(S1,S1),disp('若为 1 则说明两串完全相同,为 0
则不同.')
```

两串比较的结果是：

```
Result =
1
```

若为 1 则说明两串完全相同，为 0 则不同。除了上面介绍的这些字符串操作函数外，相关的函数还有很多，限于篇幅，不再一一介绍，有需要时可通过 MATLAB 帮助获得相关主题的信息。

1.5.3 二维字符数组

二维字符数组其实就是由字符串纵向排列构成的数组。借用构造数值数组的方法，可以用直接输入法生成或用连接函数法获得。下面用两个实例加以说明。

【例 1-54】 将 S1、S2、S3、S4 分别视为数组的 4 行，用直接输入法沿纵向构造二维字符数组。

解：输入命令后运行结果如下：

```
>> S1 = '路修远以多艰兮,';
>> S2 = '腾众车使径待.';
>> S3 = '路不周以左转兮,';
>> S4 = '指西海以为期!';
>> S = [S1;S2,' ';S3;S4,' ']        % 此法要求每行字符数相同,不够时要补齐空格
S =
路修远以多艰兮,
腾众车使径待.
路不周以左转兮,
指西海以为期!
>> S = [S1;S2,' ';S3;S4]            % 每行字符数不同时,系统提示出错
Error using vertcat
CAT arguments dimensions are not consistent.
```

可以将字符串连接生成二维数组的函数有多个，下面主要介绍 char()、strvcat() 和 str2mat() 这 3 个函数。

【例 1-55】 用 char()、strvcat() 和 str2mat() 函数生成二维字符数组的示例。

解：输入命令后运行结果如下：

```
>> S1a = 'I''m nobody,'; S1b = ' who are you?';   % 注意串中有单引号时的处理方法
>> S2 = 'Are you nobody too?';
>> S3 = 'Then there''s a pair of us.';           % 注意串中有单引号时的处理方法
>> SS1 = char([S1a,S1b],S2,S3) SS1  =
I'm nobody, who are you? Are you nobody too?
Then there's a pair of us.
>> SS2 = strvcat(strcat(S1a,S1b),S2,S3) SS2  =
I'm nobody, who are you? Are you nobody too?
Then there's a pair of us.
>> SS3 = str2mat(strcat(S1a,S1b),S2,S3) SS3  =
```

```
I'm nobody, who are you? Are you nobody too?
Then there's a pair of us.
```

例 1-55 中，strcat()和 strvcat()两函数的区别在于：前者是将字符串沿横向连接成更长的字符串，而后者是将字符串沿纵向连接成二维字符数组。

1.6 符号

MATLAB 不仅在数值计算功能方面相当出色，而且在符号运算方面也提供了专门的符号数学工具箱(symbolic math toolbox)——MuPAD Notebook。

符号数学工具箱是操作和解决符号表达式的符号函数集合，其主要功能包括符号表达式与符号矩阵的基本操作、符号微积分运算以及求解代数方程和微分方程。

符号运算与数值运算的主要区别在于：数值运算必须先对变量赋值才能进行运算；符号运算无须事先对变量进行赋值，运算结果直接以符号形式输出。

1.6.1 符号表达式的生成

在符号运算中，数字、函数、算子和变量都是以字符的形式保存并进行运算的。符号表达式包括符号函数和符号方程，两者的区别在于前者不包括等号，后者必须带等号，但它们的创建方式是相同的。

MATLAB 中创建符号表达式的方法有两种：一种是直接使用字符串变量的生成方法对其进行赋值；另一种是根据 MATLAB 提供的符号变量定义函数 sym 和 syms。

sym 函数用来定义单个符号量，调用格式为

```
符号量名 = sym('符号字符串')
```

其中符号字符串可以是常量、变量、函数或表达式。

syms 函数用来建立多个符号变量，调用格式为

```
syms 符号量名 1 符号量名 2 … 符号量名 n
```

此时变量名不需加字符串分界符(‘’)，变量间用空格分隔。

【例 1-56】 符号表达式的生成。

在 MATLAB 命令窗口输入命令：

```
clear all
clc,
y1 = 'exp(x)';                   % 直接创建符号函数
equ = 'a * x ^ 2 + b * x + c = 0';   % 直接创建符号方程
y2 = sym('exp(x)');              % 使用 sym 函数生成符号表达式
syms x y                         % 建立符号变量 x、y
y3 = x ^ 2 + y ^ 2;              % 生成符号表达式
```

运行程序输出结果如下：

```
y1 =
    exp(x)
equ =
    a*x^2+b*x+c=0
y2 =
    exp(x)
y3 =
    x^2 + y^2
```

1.6.2　符号矩阵

符号矩阵也是一种特殊的符号表达式。MATLAB中的符号矩阵也可以通过sym函数来建立，矩阵的元素可以是任何不带等号的符号表达式，其调用格式为

```
符号矩阵名=sym('符号字符串矩阵')
```

符号字符串矩阵的各元素之间可用空格或逗号分隔。

在MATALB命令行窗口输入：

```
A=sym('[aa,bb;1,a+2*b]')
```

运行程序输出结果如下：

```
A =
     [ aa,        bb]
     [  1, a + 2*b]
```

输入如下：

```
clc,clear,close all
A=sym('[a,b;1,a+2*b,1,2;4,5]')
```

运行程序输出结果如下：

```
A =
     [ a,         b, 0, 0]
     [ 1, a + 2*b, 1, 2]
     [ 4,         5, 0, 0]
```

从输出结果可以看出，与数值矩阵输出形式不同，符号矩阵的每一行两端都有方括号。

在MATLAB中，数值矩阵不能直接参与符号运算，必须先转换为符号矩阵，同样也是通过sym函数来转换。

符号矩阵也是一种矩阵，因此之前介绍的矩阵的相关运算也适用于符号矩阵。很多

应用于数值矩阵运算的函数，如 det()、inv()、rank()、eig()、diag()、triu()、tril()等，也能应用于符号矩阵。

符号矩阵的逆为

```
>> inv(A)
ans =
[ (a + 2 * b)/(a * aa - bb + 2 * aa * b), - bb/(a * aa - bb + 2 * aa * b)]
[     - 1/(a * aa - bb + 2 * aa * b), aa/(a * aa - bb + 2 * aa * b)]
```

符号矩阵的秩为

```
>> rank(A)
ans =
    2
```

符号矩阵的上三角为

```
>> triu(A)
ans =
     [ aa,       bb]
     [ 0, a + 2 * b]
```

符号矩阵的下三角为

```
>> tril(A)
ans =
     [ aa,        0]
     [ 1, a + 2 * b]
```

1.6.3 常用符号运算

符号数学工具箱中提供了符号矩阵因式分解、展开、合并、简化和通分等符号操作函数，如表 1-15 所示。

表 1-15 常用的符号运算函数

函数名	说 明	函数名	说 明
factor	符号矩阵因式分解	expand	符号矩阵展开
collect	符号矩阵合并同类项	simplify	应用函数规则对符号矩阵进行化简
simple	调用 MATLAB 其他函数对符号矩阵进行综合化简，并显示化简过程	numden	分式通分
compose	复合函数运算	finverse	反函数运算
limit	计算符号表达式极限	int	符号积分(定积分或不定积分)
diff	微分和差分函数	gradient	近似梯度函数
jiacobian	计算多元函数的 Jacobi 矩阵		

由于微积分是大学教学、科研及工程应用中最重要的基础内容之一，这里只对符号微积分运算进行举例说明，其余的符号函数运算，读者可以通过查阅 MATLAB 的帮助文档进行学习。

【例 1-57】 符号微积分运算。

解：在 MATLAB 输入以下代码：

```
>> syms t x y                    % 定义符号变量
>> f1 = sin(2 * x);
>> df1 = diff(f1)                % 对函数 f1 中变量 x 求导
```

运行程序输出结果如下：

```
df1 =
    2 * cos(2 * x)
```

输入命令如下：

```
>> f2 = x^2 + y^2;
>> df2 = diff(f2,x)              % 对函数 f2 中变量 x 求偏导
```

运行程序输出结果如下：

```
df2 =
    2 * x
```

输入命令如下：

```
>> f3 = x * sin(x * t);
>> int1 = int(f3,x)              % 求函数 f3 的不定积分
```

运行程序输出结果如下：

```
int1 =
     (sin(t * x) - t * x * cos(t * x))/t^2
```

输入命令如下：

```
>> int2 = int(f3,x,0,pi/2) % 求 f3 在[0,pi/2]区间上的定积分
```

运行程序输出结果如下：

```
int2 =
     (sin((pi * t)/2) - (pi * t * cos((pi * t)/2))/2)/t^2
```

1.7 关系运算和逻辑运算

MATLAB中运算包括算术运算、关系运算和逻辑运算。而在程序设计中应用十分广泛的是关系运算和逻辑运算。关系运算是用于比较两个操作数,而逻辑运算则是对简单逻辑表达式进行复合运算。关系运算和逻辑运算的返回结果都是逻辑类型(1代表逻辑真,0代表逻辑假)。

1.7.1 关系运算

在程序中经常需要比较两个量的大小关系,以决定程序下一步的工作。比较两个量的运算符称为关系运算符。MATLAB中的关系运算符如表1-16所示。

表1-16 关系运算符

关系运算符	说　明
<	小于
<=	小于等于
>	大于
>=	大于等于
==	恒等于
~=	不等于

当操作数是数组形式时,关系运算符总是对被比较的两个数组的各个对应元素进行比较,因此要求被比较的数组必须具有相同的尺寸。

【例1-58】 MATLAB中的关系运算。

解:在命令窗口输入:

```
>> 5 >= 4
ans =
     1
>> x = rand(1,4)
x =
    0.8147   0.9058   0.1270   0.9134
>> y = rand(1,4)
y =
    0.6324   0.0975   0.2785   0.5469
>> x > y

ans =
     1    1    0    1
```

注意:(1)比较两个数是否相等的关系运算符是两个等号"= =",而单个等号"="在MATLAB中是变量赋值的符号。

(2)比较两个浮点数是否相等时需要注意,由于浮点数的存储形式决定相对误差的

存在，在程序设计中最好不要直接比较两个浮点数是否相等，而是采用大于、小于的比较运算将待确定值限制在一个满足需要的区间之内。

1.7.2 逻辑运算

关系运算返回的结果是逻辑类型(逻辑真或逻辑假)，这些简单的逻辑数据可以通过逻辑运算符组成复杂的逻辑表达式，这在程序设计中经常用于进行分支选择或者确定循环终止条件。

MATLAB 中的逻辑运算有以下 3 类：

(1) 逐个元素的逻辑运算；

(2) 捷径逻辑运算；

(3) 逐位逻辑运算。

只有前两种逻辑运算返回逻辑类型的结果。

1. 逐个元素的逻辑运算

逐个元素的逻辑运算符有 3 种：逻辑与(&)、逻辑或(|)和逻辑非(～)。前两个是双目运算符，必须有两个操作数参与运算；逻辑非是单目运算符，只能对单个元素进行运算。其意义和示例如表 1-17 所示。

表 1-17 逐个元素的逻辑运算符

运算符	说　明	举　例
&	逻辑与：双目逻辑运算符 参与运算的两个元素值为逻辑真或非零时，返回逻辑真，否则非返回逻辑假	1&0 返回 0 1&false 返回 0 1&1 返回 1
|	逻辑或：双目逻辑运算符 参与运算的两个元素都为逻辑假或零时，返回逻辑假，否则返回逻辑真	1|0 返回 1 1|false 返回 1 0|0 返回 0
～	逻辑非：单目逻辑运算符 参与运算的元素为逻辑真或非零时，返回逻辑假，否则返回逻辑真	～1 返回 0 ～0 返回 1

注意：这里逻辑与和逻辑非运算，都是逐个元素进行双目运算，因此如果参与运算的是数组，就要求两个数组具有相同的尺寸。

【例 1-59】 逐个元素的逻辑运算。

解：在命令窗口输入：

```
>> x = rand(1,3)
x =
    0.9575   0.9649   0.1576
>> y = x > 0.5
y =
    1   1   0
>> m = x < 0.96
```

```
m =
     1   0   1
>> y&m
ans =
     1   0   0
>> y|m
ans =
     1   1   1
>> ~y
ans =
     0   0   1
```

2. 捷径逻辑运算

MATLAB 中捷径逻辑运算符有两个：逻辑与(&&)和逻辑或(||)。实际上它们的运算功能和前面讲过的逐个元素的逻辑运算符相似，只不过在一些特殊情况下，捷径逻辑运算符会减少一些逻辑判断的操作。

当参与逻辑与运算的两个数据都同为逻辑真(非零)时，逻辑与运算才返回逻辑真(1)，否则都返回逻辑假(0)。

&& 运算符就是利用这一特点，当参与运算的第一个操作数为逻辑假时，直接返回假，而不再去计算第二个操作数。

& 运算符在任何情况下都要计算两个操作数的结果，然后再逻辑与。

|| 运算符的情况类似，当第一个操作数为逻辑真时，|| 运算符直接返回逻辑真，而不再去计算第二个操作数。

| 运算符在任何情况下都要计算两个操作数的结果，然后再逻辑或。

捷径逻辑运算符如表 1-18 所示。

表 1-18　捷径逻辑运算符

运算符	说　　明
&&	逻辑与：当第一个操作数为假，直接返回假，否则同 &
||	逻辑或：当第一个操作数为真，直接返回真，否则同 |

因此，捷径逻辑运算符比相应的逐个元素的逻辑运算符的运算效率更高，在实际编程中，一般都用捷径逻辑运算符。

【例 1-60】 捷径逻辑运算。

解：在 MATLAB 命令窗口中输入以下命令：

```
>> x = 0
x =
     0
>> x~ = 0&&(1/x > 2)
ans =
     0
>> x~ = 0&(1/x > 2)
ans =
     0
```

3. 逐位逻辑运算

逐位逻辑运算能够对非负整数二进制形式进行逐位逻辑运算，并将逐位运算后的二进制数值转换成十进制数值输出。MATLAB 中逐位逻辑运算函数如表 1-19 所示。

表 1-19　逐位逻辑运算函数

函　　数	说　　明
bitand(a,b)	逐位逻辑与：a 和 b 的二进制数位上都为 1 则返回 1，否则返回 0，并将逐位逻辑运算后的二进制数值转换成十进制数值输出
bitor(a,b)	逐位逻辑或：a 和 b 的二进制数位上都为 0 则返回 0，否则返回 1，并将逐位逻辑运算后的二进制数值转换成十进制数值输出
bitcmp(a,b)	逐位逻辑非：将数字 a 扩展成 n 位二进制形式，当扩展后的二进制数位上都为 1 则返回 0，否则返回 1，并将逐位逻辑运算后的二进制数值转换成十进制数值输出
bitxor(a,b)	逐位逻辑异或：a 和 b 的二进制数位上相同则返回 0，否则返回 1，并将逐位逻辑运算后的二进制数值转换成十进制数值输出

【例 1-61】 逐位逻辑运算函数。

解：在命令窗口中输入：

```
>> m = 8;n = 2;
>> mm = bitxor(m,n);
>> dec2bin(m)
ans =
    1000
>> dec2bin(n)
ans =
    10
>> dec2bin(mm)
ans =
    1010
```

1.7.3　常用函数

除了上面的关系与逻辑运算操作符之外，MATLAB 还提供了大量的其他关系与逻辑函数，具体如表 1-20 所示。

表 1-20　关系与逻辑操作函数

函　　数	说　　明
xor(x,y)	异或运算：x 或 y 非零(真)返回 1，x 和 y 都是零(假)或都是非零(真)返回 0
any(x)	如果在一个向量 x 中，任何元素非零，返回 1；矩阵 x 中的每一列有非零元素，返回 1
all(x)	如果在一个向量 x 中，所有元素非零，返回 1；矩阵 x 中的每一列所有元素非零，返回 1

【例 1-62】 关系与逻辑操作函数的应用。

解：在 MATLAB 输入以下代码：

```
>> A = [0 0 3;0 3 3]
>> B = [0  - 2 0;1  - 2 0]
>> C = xor(A,B)
>> D = any(A)
>> E = all(A)
```

得到结果如下：

```
>>
A =
     0     0     3
     0     3     3
B =
     0    - 2     0
     1    - 2     0
C =
     0     1     1
     1     0     1
D =
     0     1     1
E =
     0     0     1
```

除了这些函数，MATLAB还提供了大量的函数用来测试特殊值或条件的存在，并返回逻辑值，如表1-21所示。

表 1-21 测试函数

函　　数	说　　明
finite	元素有限，返回真值
isempty	参量为空，返回真值
isglobal	参量是一个全局变量，返回真值
ishold	当前绘图保持状态是“ON”，返回真值
isieee	计算机执行IEEE算术运算，返回真值
isinf	元素无穷大，返回真值
isletter	元素为字母，返回真值
isnan	元素为不定值，返回真值
isreal	参量无虚部，返回真值
isspace	元素为空格字符，返回真值
isstr	参量为一个字符串，返回真值
isstudent	MATLAB为学生版，返回真值
isunix	计算机为UNIX系统，返回真值
isvms	计算机为VMS系统，返回真值

1.8 复数

复数运算从根本上讲是对实数运算的拓展，在自动控制、电路学科等自然科学与工程技术中复数的应用非常广泛。

1.8.1 复数和复矩阵的生成

复数有两种表示方式：一般形式和复指数形式。

一般形式为 $x=a+b\mathrm{i}$，其中 a 为实部，b 为虚部，i 为虚数单位。在 MATLAB 中，使用如下赋值语句：

```
>> syms a b
>> x = a + b * i
x =
    a + b * i
```

其中 a、b 为任意实数，即可生成复数 x。

复指数形式为 $x=r\cdot \mathrm{e}^{\mathrm{i}\theta}$，其中 r 为复数的模，θ 为复数的幅角，i 为虚数单位。在 MATLAB 中，使用如下赋值语句：

```
>> syms r theta
>> x = r * exp(theta * i)
x =
    r * exp(theta * i)
```

其中 r、theta 为任意实数，即可生成复数 x。

选取合适的表示方式能够便于复数运算。一般形式适合处理复数的代数运算，复指数形式适合处理复数旋转等涉及幅角改变的问题。

复数的生成有两种方法：一种是直接赋值，如上所述；另一种是通过符号函数 syms 来构造，将复数的实部和虚部看作自变量，用 subs 函数对实部和虚部进行赋值。

【例 1-63】 复数的生成。

在 MATLAB 命令窗口输入命令：

```
clear all
clc,
x1 = - 1 + 2i                % 直接赋值
x2 = sqrt(2) * exp(i * pi/4)
syms a b real
x3 = a + b * i               % 构造符号函数
subs(x3,{a,b},{ - 1,2})      % 使用 subs 函数对实部虚部赋值
```

运行程序输出结果如下：

```
x1 =
   - 1.0000 + 2.0000i
x2 =
    1.0000 + 1.0000i
x3 =
a + b * i
ans =
     - 1 + 2 * i
```

输入命令：

```
clear all
clc,
syms r theta real
x4 = r * exp(theta * i);
subs(x4,{r,theta},{sqrt(20),pi/8})
```

运行程序输出结果如下：

```
ans =
    2 * 5 ^(1/2) * ((2 ^(1/2)  + 2)^(1/2)/2  +  ((2  -  2 ^(1/2))^(1/2) * i)/2)
```

复数矩阵的生成也有两种方法：一种直接输入复数元素生成；另一种将实部和虚部矩阵分开建立，再写成和的形式，此时实部矩阵和虚部矩阵的维度必须相同。

【例 1-64】 复数矩阵的生成。

解：在 MATLAB 输入以下代码：

```
clear all
clc
A = [ - 1 + 20i, - 3 + 40i;1 - 20i,30 - 4i]   % 复数元素
```

运行程序输出结果如下：

```
A  =
   - 1.0000  + 20.0000i   - 3.0000  + 40.0000i
     1.0000  - 20.0000i     30.0000  -  4.0000i
```

矩阵 A 的实部矩阵：

```
real(A)
```

运行程序输出结果如下：

```
>> real(A)
ans  =
     - 1   - 3
       1     30
```

矩阵 A 的虚部矩阵：

```
imag(A)
```

运行程序输出结果如下：

```
>> imag(A)
ans  =
     20     40
   - 20   - 4
```

由矩阵 A 的实部和虚部构造复向量矩阵如下：

```
B = real(A);
C = imag(A);
D = B + C * i
```

运行程序输出结果如下：

```
D =
  -1.0000 + 20.0000i   -3.0000 + 40.0000i
   1.0000 - 20.0000i   30.0000 - 4.0000i
```

1.8.2 复数的运算

复数的基本运算与实数相同，都是使用相同的运算符或函数。此外，MATLAB 还提供了一些专门用于复数运算的函数，如表 1-22 所示。

表 1-22 复数运算函数

函数名	说　明	函数名	说　明
abs	求复数或复数矩阵的模	angle	求复数或复数矩阵的幅角，单位为弧度
real	求复数或复数矩阵的实部	imag	求复数或复数矩阵的虚部
conj	求复数或复数矩阵的共轭	isreal	判断是否为实数
unwrap	去掉幅角突变	cplxpair	按复数共轭对排序元素群

1.9 数据类型间的转换

MATLAB 支持不同数据类型间的转换，这给数据处理带来极大方便。常用的数据类型转换函数如表 1-23 所示。

表 1-23 数据类型转换函数

函数名	说　明	函数名	说　明
int2str	整数→字符串	dec2hex	十进制数→十六进制数
mat2str	矩阵→字符串	hex2dec	十六进制数→十进制数
num2str	数字→字符串	hex2num	十六进制数→双精度浮点数
str2num	字符串→数字	num2hex	浮点数→十六进制数
base2dec	B 底字符串→十进制数	cell2mat	元胞数组→数值数组
bin2dec	二进制数→十进制数	cell2struct	元胞数组→结构体数组
dec2base	十进制数→B 底字符串	mat2cell	数值数组→元胞数组
dec2bin	十进制数→二进制数	struct2cell	结构体数组→元胞数组

【例 1-65】 数据类型之间的切换有较多的应用，例如对于图像本身而言，图像读入的多位 uint8 型数据，需要转换成 double 型数据进行处理。

解：在 MATLAB 输入以下代码：

```
clear all
clc
im = imread('cameraman.tif');
imshow(im)
im1 = im2double(im);
imshow(im)
```

运行结果如图 1-2 所示。



图 1-2　数据类型转换

字符型变量转换，命令如下：

```
clear all
clc
a = '2'
b = double(a)
b1 = str2num(a)
c = 2 * a
d = 2 * b
d = 2 * b1
```

运行程序输出结果如下：

```
a =
2
b =
      50
b1 =
      2
c =
      100
d =
      100
d =
      4
```

本章小结

MATLAB的优化算法包括多方面,涵盖面极广。本章主要围绕向量、矩阵、字符串、符号、关系运算和逻辑运算、复数等内容进行介绍,通过本章基础知识学习,用户可以根据自身需求,进行简单的优化程序的编写。

第2章 MATLAB编程

MATLAB语言称为第四代编程语言，程序简洁，可读性强而且调试十分容易，是MATLAB的重要组成部分。MATLAB为用户提供了非常方便易懂的程序设计方法，类似于其他的高级语言编程。

本章侧重于MATLAB中最基础的程序设计，分别介绍编程原则、分支结构、循环结构、其他控制程序命令及程序调试等内容。

学习目标：

(1) 了解MATLAB编程；

(2) 掌握MATLAB编程原则；

(3) 掌握MATLAB各种控制指令；

(4) 熟悉MATLAB程序的调试。

2.1 MATLAB编程概述

计算机程序就是计算机指令的集合，不同的编程语言指令与功能是不一样的。MATLAB语言是一种面向对象的高级语言，它具有编程效率高、易学易用的优点。

MATLAB语言是当前很多领域的首选计算机语言，也是最适合众多理工科专业的计算机数学语言。

MATLAB中各种命令可以完成许多单一的任务，对于某些较为复杂的问题，仅靠现有的命令或函数解决，往往难以直接达到目的。为此，要运用MATLAB编程语言编制程序，形成M文件。MATLAB中每一个命令都是一个M文件。

MATLAB程序(M文件)有两种形式：一种是可直接运行的命令文件；另一种是可供调用的函数文件。这两种文件的扩展名相同，均为“.m”，故称为M文件。

MATLAB程序编辑是在编辑器中进行，程序运行结果或错误信息显示在命令行窗口，程序运行过程产生的参数信息显示在工作区，具体如图2-1所示。因为程序有问题，图2-1的命令行窗口中出现了程序错误提示。

【例2-1】 修改图2-1中程序，使得命令行窗口无错误提示，并给

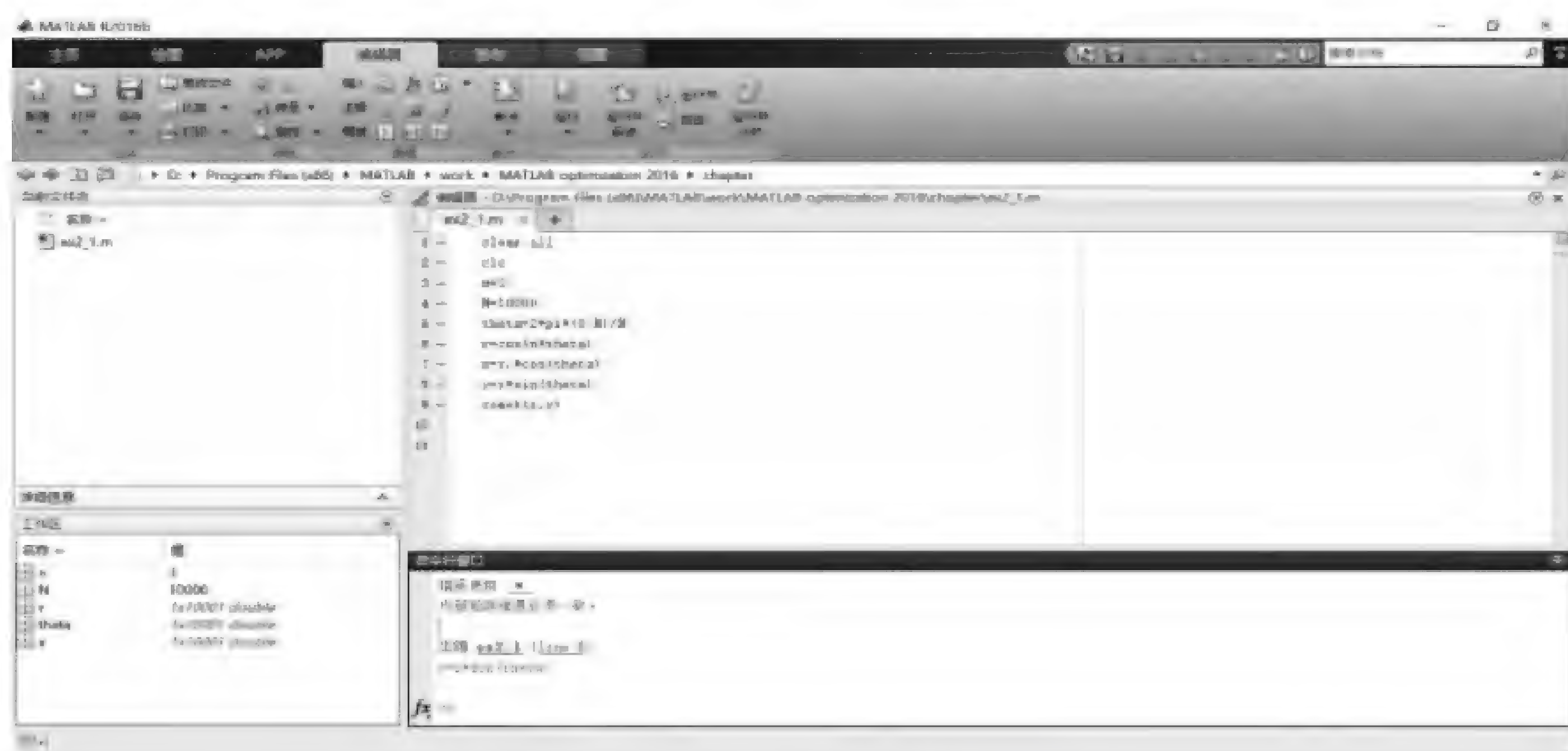


图 2-1　MATLAB 编辑器错误提示

出正确结果。

解：图 2-1 中第 8 行中乘号运用有问题，修改如下：

```
clear all
clc
n = 3;
N = 10000;
theta = 2 * pi * (0:N)/N;
r = cos(n * theta);
x = r. * cos(theta);
y = r. * sin(theta);
comet(x,y)
```

运行程序后，得到优化后的程序运行界面如图 2-2 所示，正确结果如图 2-3 所示。

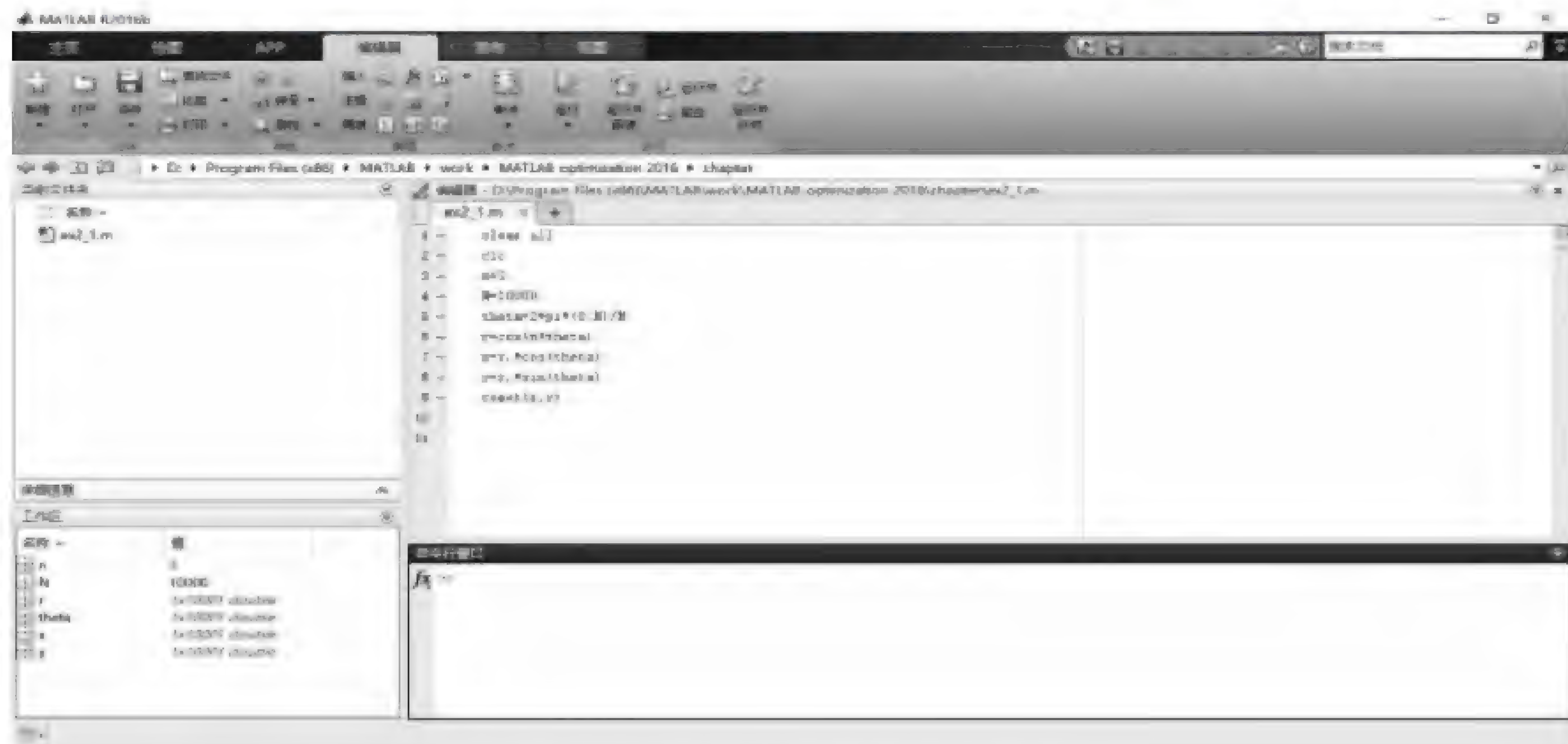


图 2-2　优化后的程序运行界面

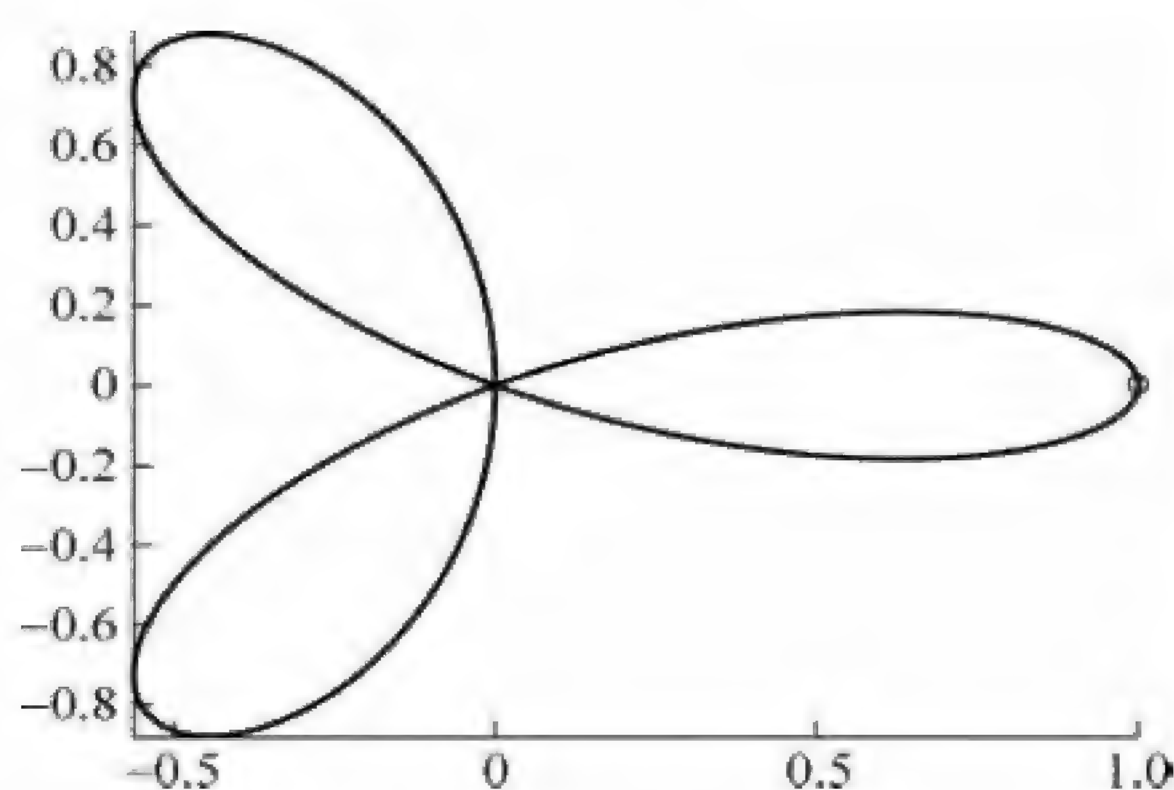


图 2-3　程序输出的正确结果

2.2　MATLAB 编程原则

MATLAB 代码的建议通常强调的是效率，例如有关“不要用循环”等的建议。除此之外，还要考虑代码(格式)的正确性、清晰性与通用性，具体含义如下：

(1) 正确：程序能准确地实现原有仿真目的；

(2) 高效：循环向量化，少用或不用循环，尽量调用 MATLAB 自带函数；

(3) 清晰：养成良好的编程习惯，使程序具有良好的可读性；

(4) 通用：程序具有高度的可移植性和可扩展性，便于后续开发调用。

在 MATLAB 编程中，还需要遵循以下几个规则：

(1) 定义变量，以英文单词小写缩写开头表示类别名，再接具体变量的英文名称。

例如，定义变量存储临时数组 TempArray 的最大值 maxTempArray。

根据工程大小确定变量名长短，小范围应用的变量应该用短的变量名。定义要清晰，避免混淆。

(2) 循环变量使用常用变量 i、j、k。程序中使用复数时，采用 i、j 以外的循环变量以避免和虚数单位冲突，同时要在注释部分说明变量的含义。

(3) 编写的程序应该高内聚、低耦合、模块函数化，且便于移植、重复使用。

(4) 使用 if 语句判断变量是否等于某一常数时，常将变量数写在等号之前，常数写在等号之后。

例如判断变量 a 是否等于 10，写为 if a==10。

(5) 用常数代替数字，少用或不用数字。

例如第 4 条示例中，如果要定义期望常量，写为 if a==10 则不标准。应该先定义 meanConst=10，同时在注释中说明，然后在程序部分写为 if a==const。如果后续要修改期望常量，则在程序定义部分修改。

2.3　分支结构

MATLAB 程序结构一般可分为顺序结构、循环结构、分支结构 3 种。顺序结构是指按顺序逐条执行，循环结构与分支结构都有其特定的语句，这样可以增强程序的可读性。

在 MATLAB 中常用的分支程序结构包括 if 结构和 switch 结构。

2.3.1 if 分支结构

如果在程序中需要根据一定条件来执行不同的操作时，可以使用条件语句，在 MATLAB 中提供 if 分支结构，或者称为 if-else-end 语句。

根据不同的条件情况，if 分支结构有多种形式，其中最简单的用法是，如果条件表达式为真，则执行语句 1，否则跳过该组命令。

if 结构是一个条件分支语句，若满足表达式的条件，则往下执行；若不满足，则跳出 if 结构。else if 表达式 2 与 else 为可选项，这两条语句可依据具体情况取舍。

if 语法结构如下：

```
if 表达式 1
    语句 1
    else if 表达式 2 (可选)
        语句 2
    else (可选)
        语句 3
    end
end
```

注意：(1) 每一个 if 都对应一个 end，即有几个 if，就应有几个 end；

(2) if 分支结构是所有程序结构中最灵活的结构之一，可以使用任意多个 else if 语句，但是只能有一个 if 语句和一个 end 语句；

(3) if 语句可以相互嵌套，可以根据实际需要将各个 if 语句进行嵌套，从而解决比较复杂的实际问题。

【例 2-2】 思考下列程序及其运行结果，说明原因。

解：在 MATLAB 命令窗口中输入以下程序：

```
clear all
clc
a = 100;
b = 20;
if a < b
   fprintf ('b>a')        % 在 Word 中输入'b>a'单引号不可用,要在 Editor 中输入
else
   fprintf ('a>b')        % 在 Word 中输入'b>a'单引号不可用,要在 Editor 中输入
end
```

运行后得到：

```
a > b
```

在程序中用到了 if-else-end 的结构，如果 a<b，则输出 b>a；反之，输出 a>b。由于

a=100,b=20,比较可得结果 a>b。

在分支结构中,多条语句可以放在同一行,但语句间要用“;”分开。

2.3.2 switch 分支结构

和 C 语言中的 switch 分支结构类似,在 MATLAB 中适用于条件多而且比较单一的情况,类似于一个数控的多个开关。其一般的语法调用方式如下:

```
switch 表达式
case 常量表达式 1
    语句组 1
    case 常量表达式 2
        语句组 2
        …
    otherwise
        语句组 n
end
```

其中,switch 后面的表达式可以是任何类型,如数字、字符串等。

当表达式的值与 case 后面常量表达式的值相等时,就执行这个 case 后面的语句组,如果所有的常量表达式的值都与这个表达式的值不相等,则执行 otherwise 后面的语句组。

表达式的值可以重复,在语法上并不错误,但是在执行时,后面符合条件的 case 语句将被忽略。

各个 case 和 otherwise 语句的顺序可以互换。

【例 2-3】 输入一个数,判断它能否被 5 整除。

解:在 MATLAB 中输入以下程序:

```
clear all
clc
n = input('输入 n = ');          % 输入 n 值
switch mod(n,5)                 % mod 是求余函数,余数为 0,得 0,余数不为 0,得 1
case 0
   fprintf ('%d 是 5 的倍数',n)
otherwise
   fprintf('%d 不是 5 的倍数',n)
end
```

运行后结果为

```
输入 n = 12
12 不是 5 的倍数>>
```

在 switch 分支结构中,case 命令后的检测不仅可以为一个标量或字符串,还可以为一个元胞数组。如果检测值是一个元胞数组,MATLAB 将把表达式的值和该元胞数组

中的所有元素进行比较,如果元胞数组中某个元素和表达式的值相等,MATLAB 认为比较结果为真。

2.4 循环结构

在 MATLAB 程序中,循环结构主要包括 while 循环结构和 for 循环结构两种。下面对两种循环结构作详细介绍。

2.4.1 while 循环结构

除了分支结构之外,MATLAB 还提供多个循环结构。和其他编程语言类似,循环语句一般用于有规律的重复计算。被重复执行的语句称为循环体,控制循环语句流程的语句称为循环条件。

在 MATLAB 中,while 循环结构的语法形式如下:

```
while 逻辑表达式
    循环语句
end
```

while 结构依据逻辑表达式的值判断是否执行循环体语句。若表达式的值为真,则执行循环体语句一次,在反复执行时,每次都要进行判断。若表达式为假,则程序执行 end 之后的语句。

为了避免因逻辑上的错误,而陷入死循环,建议在循环体语句的适当位置加 break 语句,以便程序能正常执行。

while 循环也可以嵌套,其结构如下:

```
while 逻辑表达式 1
    循环体语句 1
while 逻辑表达式 2
    循环体语句 2
end
循环体语句 3
end
```

【例 2-4】 请设计一段程序,求 1~100 的偶数和。

解:在 MATLAB 命令窗口输入以下程序:

```
clear all
clc
x = 0;              % 初始化变量 x
sum = 0;            % 初始化 sum 变量
   while x < 101    % 当 x < 101 执行循环体语句
     sum = sum + x; % 进行累加
```

```
      x = x + 2;
   end            % while 结构的终点
sum               % 显示 sum
```

运行后的结果为

```
sum =
        2550
```

【例 2-5】 请设计一段程序，求 1～100 的奇数和。

解：在 MATLAB 命令窗口输入以下程序：

```
clear all
clc
x = 1;              % 初始化变量 x
sum = 0;            % 初始化 sum 变量
   while x < 101    % 当 x < 101 执行循环体语句
      sum = sum + x; % 进行累加
      x = x + 2;
   end              % while 结构的终点
sum                 % 显示 sum
```

运行后的结果为

```
sum =
        2500
```

2.4.2 for 循环结构

在 MATLAB 中，另外一种常见的循环结构是 for 循环，常用于知道循环次数的情况，其语法规则如下：

```
for ii = 初值: 增量: 终值
    语句 1
    …
    语句 n
end
```

ii＝初值：终值，则增量为 1。初值、增量、终值可正可负，可以是整数，也可以是小数，只须符合数学逻辑。

【例 2-6】 请设计一段程序，求 1＋2＋…＋100 的和。

解：程序设计如下：

```
clear all
clc
sum = 0;                    % 设置初值(必须要有)
for ii = 1:100;             % for 循环,增量为 1
    sum = sum + ii;
end
sum
% end                       % 程序结束
```

运行后的结果为

```
sum =
        5050
```

【例 2-7】 比较以下两个程序的区别。

解：MATLAB 程序 1 设计如下：

```
clear all
clc
for ii = 1:100;             % for 循环,增量为 1
    sum = sum + ii;
end
sum
% end                       % 程序结束
```

运行后的结果为

```
sum =
       10100
```

程序 2 设计如下：

```
>> clear all
clc
for ii = 1:100;              % for 循环,增量为 1
sum = sum + ii;
end
sum
% end                        % 程序结束
```

运行后的结果为

```
??? Error: "sum" was previously used as a function,
conflicting with its use here as the name of a variable.
```

一般的高级语言中，变量若没有设置初值，程序会以 0 作为其初始值，然而这在 MATLAB 中是不允许的。所以，在 MATLAB 中，应给出变量的初值。

程序1没有clear，则程序可能会调用到内存中已经存在的sum值，其结果就成了sum=10100。

在程序2中与上一题的差别是少了sum=0这条语句，出现这种情况时，因为程序中有clear语句，则显示错误信息。

注意：while循环和for循环都是比较常见的循环结构，但是两个循环结构还是有区别的。其中最明显的区别在于，while循环的执行次数是不确定的，而for循环的执行次数是确定的。

2.5 其他控制程序命令

在使用MATLAB设计程序时，经常遇到提前终止循环、跳出子程序、显示错误等情况，因此需要其他的控制语句来实现上面的功能。在MATLAB中，对应的控制语句有continue、break、return等。

1. continue命令

continue语句通常用于for或while循环体中，其作用就是终止一趟语句的执行，也就是说它可以跳过本趟循环中未被执行的语句，去执行下一轮的循环。下面使用一个简单的实例，说明continue命令的使用方法。

【例2-8】 请思考下列程序及其运行结果，说明原因。

解：在MATLAB中输入以下程序：

```
clear all
clc
a = 3;
b = 6;
for ii = 1: 3
   b = b + 1
   if ii < 2
     continue
   end              % if 语句结束
   a = a + 2
end                 % for 循环结束
 % end
```

运行程序输出结果如下：

```
b =
     7
b =
     8
a =
     5
b =
     9
a =
     7
```

当 if 条件满足时，程序将不再执行 continue 后面的语句，而是开始下一轮的循环。continue 语句常用于循环体中，与 if 一同使用。

2. break 命令

break 语句也通常用于 for 或 while 循环体中，与 if 一同使用。当 if 后的表达式为真时就调用 break 语句，跳出当前的循环。它只终止最内层的循环。

【例 2-9】 请思考下列程序及其运行结果，说明原因。

解：在 MATLAB 中输入以下程序：

```
clear all
clc
a = 3;
b = 6;
for ii = 1: 3
    b = b + 1
    if ii > 2
        break
    end
    a = a + 2
end
 % end
```

运行程序输出结果如下：

```
b =
      7
a =
      5
b =
      8
a =
      7
b =
      9
```

从以上程序可以看出，当 if 表达式的值为假时，程序执行 a=a+2；当 if 表达式的值为真时，程序执行 break 语句，跳出循环。

3. return 命令

通常情况下，当被调用函数执行完毕后，MATLAB 会自动地把控制转至主调函数或者指定窗口。如果在被调用函数中插入 return 命令，可以强制 MATLAB 结束执行该函数并把控制转出。

return 命令是终止当前命令的执行，并且立即返回到上一级调用函数或等待键盘输入命令，可以用来提前结束程序的运行。

在 MATLAB 的内置函数中，很多函数的程序代码中引入了 return 命令，下面引用

一个简要的 det 函数代码如下：

```
function d = det(A)
if isempty(A)
    a = 1;
    return
else
    ...
end
```

在上面的程序代码中，首先通过函数语句来判断函数 A 的类型，当 A 是空数组时，直接返回 a=1，然后结束程序代码。

4. input 命令

在 MATLAB 中，input 命令的功能是将 MATLAB 的控制权暂时借给用户，然后用户通过键盘输入数值、字符串或者表达式，通过按 Enter 键将输入的内容输入到工作空间中，同时将控制权交换给 MATLAB，其常用的调用格式如下：

user_entry=input('prompt')：将用户输入的内容赋给变量 user_entry。

user_entry=input('prompt','s')：将用户输入的内容作为字符串赋给变量 user_entry。

【例 2-10】 在 MATLAB 中演示如何使用 input 命令。

解： 在 MATLAB 命令窗口输入并运行以下代码：

```
clear all
clc
a = input('input a number: ')                  % 输入数值给 a
input a number: 45
a  =
    45
b = input('input a number: ','s')              % 输入字符串给 b
input a number: 45
b  =
45
input('input a number: ')                      % 将输入值进行运算
input a number: 2 + 3
ans  =
     5
```

5. keyboard 命令

在 MATLAB 中，将 keyboard 命令放置到 M 文件中，将使程序暂停运行，等待键盘命令。通过提示符 k 来显示一种特殊状态，只有当用户使用 return 命令结束输入后，控制权才交还给程序。在 M 文件中使用该命令，对程序的调试和在程序运行中修改变量都会十分方便。

【例 2-11】 在 MATLAB 中演示如何使用 keyboard 命令。

解：keyboard 命令使用过程如下：

```
>> keyboard
K>> for i = 1: 9
    if i == 3
       continue
     end
     fprintf('i = %d\n',i)
     if i == 5
        break
     end
end
i = 1
i = 2
i = 4
i = 5
K>> return
>>
```

从上面的程序代码中可以看出，当输入 keyboard 命令后，在提示符的前面会显示 k 提示符，而当用户输入 return 后，提示符恢复正常的提示效果。

在 MATLAB 中，keyboard 命令和 input 命令的不同在于：keyboard 命令用户可以输入任意多个 MATLAB 命令，而 input 命令则只能输入赋值给变量的数值。

6. error 和 warning 命令

在 MATLAB 中，编写 M 文件的时候，经常需要提示一些警告信息。为此，MATLAB 提供了下面几个常见的命令。

error('message')：显示出错信息 message，终止程序。

errordlg('errorstring', 'dlgname')：显示出错信息的对话框，对话框的标题为 dlgname。

warning('message')：显示出错信息 message，程序继续进行。

【例 2-12】 查看 MATLAB 的不同错误提示模式。

解：在 MATLAB 编辑器中输入以下程序，并将其保存为"error"文件。

```
n = input('Enter: ');
if n < 2
    error('message');
else
    n = 2;
end
```

返回 MATLAB 命令窗口，在命令窗口输入"error"，然后分别输入数值 1 和 2，得到如下结果：

```
>> error
Enter:1
Attempt to execute SCRIPT error as a function:
C: \Program Files\MATLAB\R2012a\work\8\error.m
Error in error (line 4)
    error('message');
>> error
Enter: 2
```

将上述编辑器中程序修改为如下程序：

```
n = input('Enter: ');
if n < 2
%   errordlg('Not enough input data', 'Data Error');
    warning('message');
else
    n = 2;
end
```

返回 MATLAB 命令窗口，在命令窗口输入“error”，然后分别输入数值 1 和 2，得到如下结果：

```
>> error
Enter:1
Warning: message
> In error at 4
>> error
Enter: 2
```

在上面的程序代码中，演示了 MATLAB 中不同的错误信息提示模式。其中，error 和 warning 命令的主要区别在于 warning 命令指示警告信息后继续运行程序。

2.6 程序调试

程序调试的目的是检查程序是否正确，即程序能否顺利运行并得到预期结果。在运行程序之前，应先设想到程序运行的各种情况，测试在各种情况下程序是否能正常运行。

MATLAB 程序调试工具只能对 M 文件中的语法错误和运行错误进行定位，但是无法评价该程序的性能。MATLAB 提供了一个性能剖析指令 profile，使用它可以评价程序的性能指标，获得程序各个环节的耗时分析报告。用户可以依据该分析报告寻找程序运行效率低下的原因，以便修改程序。

2.6.1 程序调试命令

MATLAB 提供了一系列程序调试命令，利用这些命令，可以在调试过程中设置、清

除和列出断点，逐行运行 M 文件，在不同的工作区检查变量，用来跟踪和控制程序的运行，帮助寻找和发现错误。所有的程序调试命令都是以字母 db 开头的，如表 2-1 所示。

表 2-1 程序调试命令

命　　令	功　　能
dbstop in fname	在 M 文件 fname 的第一可执行程序上设置断点
dbstop at r in fname	在 M 文件 fname 的第 r 行程序上设置断点
dbstop if v	当遇到条件 v 时，停止运行程序。当发生错误时，条件 v 可以是 error，当发生 NaN 或 inf 时，也可以是 naninf/infnan
dstop if warning	如果有警告，则停止运行程序
dbclear at r in fname	清除文件 fname 的第 r 行处断点
dbclear all in fname	清除文件 fname 中的所有断点
dbclear all	清除所有 M 文件中的所有断点
dbclear in fname	清除文件 fname 第一可执行程序上的所有断点
dbclear if v	清除第 v 行由 dbstop if v 设置的断点
dbstatus fname	在文件 fname 中列出所有的断点
Mdbstatus	显示存放在 dbstatus 中用分号隔开的行数信息
dbstep	运行 M 文件的下一行程序
dbstep n	执行下 n 行程序，然后停止
dbstep in	在下一个调用函数的第一可执行程序处停止运行
dbcont	执行所有行程序直至遇到下一个断点或到达文件尾
dbquit	退出调试模式

进行程序调试，要调用带有一个断点的函数。当 MATLAB 进入调试模式时，提示符为 K>>。最重要的区别在于现在能访问函数的局部变量，但不能访问 MATLAB 工作区中的变量。具体的调试技术，请读者在调试程序的过程中逐渐体会。

2.6.2 常见程序错误

1. 输入错误

常见的输入错误除了在写程序时疏忽所导致的手误外，一般还有：

(1) 在输入某些标点时没有切换成英文状态；

(2) 表循环或判断语句的关键词 for、while、if 的个数与 end 的个数不对应（尤其是在多层循环嵌套语句中）；

(3) 左右括号不对应。

2. 语法错误

不符合 MATLAB 语言的规定，即为语法错误。

例如在用 MATLAB 语句表示数学式 $k1 \leqslant x \leqslant k2$ 时，不能直接写成“k1<=x<=k2”，而应写成“k1<=x&x<=k2”。此外，输入错误也可能导致语法错误。

3. 逻辑错误

在程序设计中逻辑错误也是较为常见的一类错误，这类错误往往隐蔽性较强、不易查找。产生逻辑错误的原因通常是算法设计有误，这时需要对算法进行修改。

4. 运行错误

程序的运行错误通常包括不能正常运行和运行结果不正确，出错的原因一般有：

(1) 数据不对，即输入的数据不符合算法要求；

(2) 输入的矩阵大小不对，尤其是当输入的矩阵为一维数组时，应注意行向量与列向量在使用上的区别；

(3) 程序不完善，只能对某些数据运行正确，而对另一些数据则运行错误，或是根本无法正常运行，这有可能是算法考虑不周所致。

对于简单的 MATLAB 程序中出现的语法错误，可以采用直接调试法，即直接运行该 M 文件，MATLAB 将直接找出语法错误的类型和出现的地方，根据 MATLAB 的反馈信息对语法错误进行修改。

当 M 文件很大或 M 文件中含有复杂的嵌套时，则需要使用 MATLAB 调试器来对程序进行调试，即使用 MATLAB 提供的大量调试函数以及与之相对应的图形化工具。

下面通过一个判断 2000 年至 2010 年间的闰年年份的示例来介绍 MATLAB 调试器的使用方法。

【例 2-13】 编写一个判断 2000 年至 2010 年间的闰年年份的程序并调试。

解：(1) 创建一个 leapyear.m 的 M 函数文件，并输入如下函数程序代码：

```
%程序为判断2000年至2010年10年间的闰年年份
%本程序没有输入/输出变量
%函数的使用格式为leapyear,输出结果为2000年至2010年10年间的闰年年份
function leapyear                    %定义函数leapyear
for year = 2000: 2010                %定义循环区间
    sign = 1;
    a = rem(year,100);               %求year除以100后的剩余数
    b = rem(year,4);                 %求year除以4后的剩余数
    c = rem(year,400);               %求year除以400后的剩余数
    if a  = 0                        %以下根据a、b、c是否为0对标志变量sign进行处理
        signsign = sign - 1;
    end
    if b = 0
        signsign = sign + 1;
    end
    if c = 0
        signsign = sign + 1;
    end
    if sign = 1
        fprintf('%4d \n',year)
    end
end
```

(2) 运行以上 M 程序，此时 MATLAB 命令窗口会给出如下错误提示：

```
>> leapyear
Error: File: leapyear.m Line: 10 Column: 10
The expression to the left of the equals sign is not a valid target for an assignment.
```

由错误提示可知，在程序的第 10 行存在语法错误。检测可知 if 选择判断语句中，用户将"=="写成了"="。因此将"="改成"=="，同时也更改第 13、16、19 行中的"="为"=="。

(3) 程序修改并保存完成后，可直接运行修正后的程序，程序运行结果如下：

```
>> leapyear
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
```

显然，2001 年至 2010 年间不可能每年都是闰年，由此判断程序存在运行错误。

(4) 分析原因。可能由于在处理年号是否是 100 的倍数时，变量 sign 存在逻辑错误。

(5) 断点设置。断点为 MATLAB 程序执行时人为设置的中断点，程序运行至断点时便自动停止运行，等待用户的下一步操作。设置断点只需要用鼠标单击程序左侧的"-"，使得"-"变成红色的圆点(当存在语法错误时圆点颜色为灰色)，如图 2-4 所示。

应该在可能存在逻辑错误或需要显示相关代码执行数据的附近设置断点，例如本例中的第 12、15 和 18 行。如果用户需要去除断点，可以再次单击红色圆点去除，也可以单击工具栏中的工具去除所有断点。

(6) 运行程序。按 F5 键或单击工具栏中的按钮执行程序，这时其他调试按钮将被激活。程序运行至第一个断点时暂停，在断点右侧出现向右指向的绿色箭头，如图 2-5 所示。

程序调试运行时，在 MATLAB 的命令窗口中将显示如下内容：

```
>> leapyear
12    end
K>>
```

此时可以输入一些调试指令，更加方便对程序调试的相关中间变量进行查看。



图 2-4 断点标记



图 2-5 程序运行至断点处暂停

(7) 单步调试。可以通过按 F10 键或单击工具栏中相应的单步执行图形按钮，此时程序将一步一步按照用户需求向下执行，如图 2-6 所示，在按 F10 键后，程序从第 12 步运行到第 13 步。



图 2-6　程序单步执行

(8) 查看中间变量。可以将鼠标停留在某个变量上，MATLAB 将会自动显示该变量的当前值，如图 2-7 所示。也可以在 MATLAB 的 workspace 中直接查看所有中间变量的当前值，如图 2-8 所示。

(9) 修正代码。通过查看中间变量可知，在任何情况下 sign 的值都是 1，此时调整并修改程序代码如下：

```
>> %程序为判断 2000 年至 2010 年 10 年间的闰年年份
 %本程序没有输入/输出变量
 %函数的使用格式为 leapyear,输出结果为 2000 年至 2010 年 10 年间的闰年年份
function leapyear
for year = 2000: 2010
    sign = 0;
    a  =  rem(year,400);
    b  =  rem(year,4);
    c  =  rem(year,100);
    if a  = = 0
        sign = sign + 1;
    end
```



图 2-7　用鼠标停留方法查看中间变量

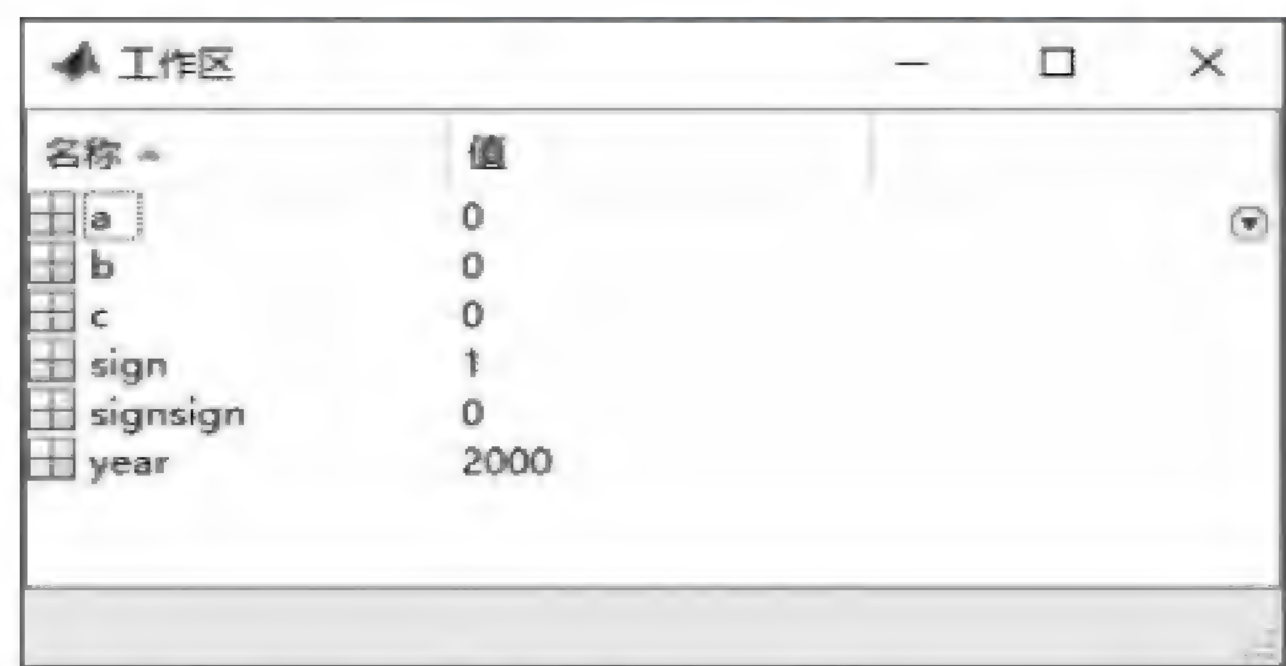


图 2-8　查看 workspace 中所有中间变量的当前值

```
    if b == 0
        sign = sign + 1;
    end
    if c == 0
        sign = sign - 1;
    end
    if sign == 1
        fprintf('%4d \n',year)
    end
end
```

按 F5 键再次执行程序，得到的运行结果如下：

```
>> leapyear
2000
2004
2008
```

分析发现，结果正确，此时程序调试结束。

2.6.3 内存优化

内存优化对于一些普通的用户而言可以不用顾及，因为随着计算机的发展，内存容量已经能够满足大多数数学运算的要求。MATLAB 本身对计算机内存优化提供的操作支持较少，只有遇到超大规模运算时，内存优化才能起到作用。下面给出几个比较常见的内存操作函数，可以在需要时使用。

(1) whos：查看当前内存使用状况的函数；

(2) clear：删除变量及其内存空间，可以减少程序的中间变量；

(3) save：将某个变量以 mat 数据文件的形式存储到磁盘中；

(4) load：载入 mat 数据到内存空间。

由于内存操作函数在函数运行时使用较少，合理的优化内存操作往往由用户编写程序时养成的习惯和经验决定，一些好的做法如下：

(1) 尽量保证创建变量的集中性，最好在函数开始时创建；

(2) 对于含零元素多的大型矩阵，尽量转化为稀疏矩阵；

(3) 及时清除占用内存很大的临时中间变量；

(4) 尽量少开辟新的内存，而是重用内存。

程序优化的本质也是算法优化。如果一个算法描述的比较详细，它几乎也就指定了程序的每一步。若算法本身描述的不够详细，在编程时会给某些步骤的实现方式留有较大空间，这样就需要找到尽可能好的实现方式以达到程序优化的目的，但一般情况下认为算法是足够详细的。如果一个算法设计的足够“优”，就等于从源头上控制了程序走向“劣质”。

算法优化的一般要求是不仅在形式上尽量做到步骤简化、简单易懂，更重要的是能用最低的时间复杂度和空间复杂度完成所需计算。这包括巧妙的设计程序流程，灵活的控制循环过程(如及时跳出循环或结束本次循环)，较好的搜索方式及正确的搜索对象等，以避免不必要的计算过程。例如在判断一个整数 m 是否是素数时，可以看它能否被 m/2 以前的整数整除。而更快的方法是，只需看它能否被 m 以前的整数整除就可以了。再如，在求两个整数之间的所有素数时跳过偶数直接对奇数进行判断，这都体现了算法优化的思想。下面通过几个具体的例子来体会其中所包含的优化思想。

【例 2-14】 编写冒泡排序算法程序。

解：冒泡排序是一种简单的交换排序，其基本思想是两两比较待排序记录中的元素，如果是逆序则进行交换，直到这个记录中没有逆序的元素。

该算法的基本操作是逐趟进行比较和交换。第一趟比较将最大记录放在 x[n]的位置。一般地,第 i 趟从 x[1]到 x[n－i＋1]依次比较相邻的两个记录,将这 n－i＋1 个记录中的最大者放在了第 n－i＋1 的位置上。其算法程序如下:

```
function s = BubbleSort(x)
%冒泡排序,x为待排序数组
n = length(x);
for i = 1: n - 1                %最多做n-1趟排序
    flag = 0;                   %flag为交换标志,本趟排序开始前,交换标志应为假
    for j = 1: n - i            %每次从前向后扫描,j从1到n-i
        if x(j)> x(j + 1)       %如果前项大于后项则进行交换
            t = x(j + 1);
            x(j + 1) = x(j);
            x(j) = t;
            flag = 1;           %当发生了交换,将交换标志置为真
        end
    end
    if (~flag)                  %若本趟排序未发生交换,则提前终止程序
        break;
    end
end
s = x;
```

本程序通过使用标志变量 flag 来标志在每一趟排序中是否发生了交换,若某趟排序中一次交换都没有发生则说明此时数组已经为有序(正序),应提前终止算法(跳出循环)。若不使用这样的标志变量来控制循环往往会增加不必要的计算量。

【例 2-15】 公交线路查询问题。设计一个查询算法,给出一个公交线路网中从起始站 s1 到终止站 s2 之间的最佳线路。其中一个最简单的情形就是查找直达线路,假设相邻公交车站的平均行驶时间(包括停站时间)为 3 分钟,若以时间最少为择优标准,请在此简化条件下完成查找直达线路的算法,并根据附录数据(见题后数据),利用此算法求出以下起始站到终止站之间的最佳路线。

(1) 242→105;(2) 117→53;(3) 179→201;(4) 16→162。

解:为了便于 MATLAB 程序计算,应先将线路信息转化为矩阵形式,导入 MATLAB(可先将原始数据经过文本导入 Excel)。每条线路可用一个一维数组来表示,且将该线路终止站以后的节点用 0 来表示,每条线路从上往下顺序排列构成矩阵 A。

此算法的核心是线路选择问题,要找最佳线路,应先找到所有的可行线路,然后再以所用的时间为关键字选出用时最少的线路。在寻找可行线路时,可先在每条线路中搜索 s1,当找到 s1 则接着在该线路中搜索 s2,若又找到 s2,则该线路为一可行线路,记录该线路及所需时间,并结束对该线路的搜索。

另外,在搜索 s1 与 s2 时若遇到 0 节点,则停止对该数组的遍历。

```
%A为线路信息矩阵,s1,s2分别为起始站和终止站
%返回值L为最佳线路,t为所需时间
[m,n] = size(A);
```

```
L1 = [];t1 = [];                    %L1 记录可行线路,t1 记录对应线路所需时间
for i = 1: m
    for j = 1: n
        if A(i,j) == s1             %若找到 s1,则从下一站点开始寻找 s2
            for k = j + 1: n
                if A(i,k) == 0      %若此节点为 0,则跳出循环
                    break;
                elseif A(i,k) == s2 %若找到 s2,记录该线路及所需时间, 然后跳出循环
                    L1 = [L1,i];
                    t1 = [t1,(k - j) * 3];
                    break;
                end
            end
        end
    end
end
m1 = length(L1);                    %测可行线路的个数
if m1 == 0                          %若没有可行线路,则返回相应信息
    L = 'No direct line';
    t = 'Null';
elseif m1 == 1
    L = L1;t = t1;              %否则,存在可行线路,用 L 存放最优线路,t 存放最小的时间 else
    L = L1(1);t = t1(1);            %分别给 L 和 t 赋初值为第一条可行线路和所需时间
    for i = 2: m1
        if t1(i)< t                 %若第 i 条可行线路的时间小于 t,
            L = i;                  %则给 L 和 t 重新赋值
            t = t1(i);
        elseif t1(i) == t           %若第 i 条可行线路的时间等于 t,
            L = [L,L1(i)];          %则将此线路并入 L
        end
    end
end
```

首先说明,这个程序能正常运行并得到正确结果。但仔细观察之后就会发现它的不足之处:一个是在对 j 的循环中应先判断节点是否为 0,若为 0 则停止向后访问,转向下一条路的搜索;另一个是对于一个二维的数组矩阵,用两层(不是两个)循环进行嵌套就可以遍历整个矩阵,得到所有需要的信息,而上面的程序中却出现了三层循环嵌套的局面。

其实,在这种情况下,倘若找到了 s2,本该停止对此线路节点的访问,但这里的 break 只能跳出对 k 的循环,而对该线路数组节点的访问(即对 j 的循环)将会一直进行到 n,做了大量的无用功。

为了消除第三层的循环能否对第二个循环内的判断语句做如下修改:

```
if A(i,j) == s1
    continue;
    if A(i,k) == s2
        L1 = [L1,i];
        t1 = [t1,(k - j) * 3];
        break;
    end
end
```

这种做法企图控制流程在搜到 s1 时能继续向下走，搜索 s2，而不用再嵌套循环。这样却是行不通的，因为即使 s1 的后面有 s2，也会被先被 if A(i,j)==s1 拦截，continue 后的语句将不被执行。所以，经过这番修改后得到的其实是一个错误的程序。

事实上，若想消除第三层循环可将这第三层循环提出来放在第二层成为与 j 并列的循环，若在对 j 的循环中找到了 s1，可用一个标志变量对其进行标志，然后再对 s1 后的节点进行访问，查找 s2。综上，可将第一个 for 循环内的语句修改如下：

```
flag = 0;                    % 用 flag 标志是否找到 s1,为其赋初值为假
for j = 1: n
    if A(i,j) == 0           % 若该节点为 0,则停止对该线路的搜索,转向下一条线路
        break;
    elseif A(i,j) == s1      % 否则,若找到 s1,置 flag 为真,并跳出循环
        flag = 1;
        break;
    end
end
if flag                      % 若 flag 为真,则找到 s1,从 s1 的下一节点开始搜索 s2
    for k = j + 1: n
        if A(i,k) == 0
            break;
        elseif A(i,k) == s2  % 若找到 s2,记录该线路及所需时间,然后跳出循环
            L1 = [L1,i];
            t1 = [t1,(k - j) * 3];
            break;
        end
    end
end
```

若将程序中重叠的部分合并还可以得到一种形式上更简洁的方案：

```
q = s1;                      % 用 q 保存 s1 的原始值
for i = 1: m
        s1 = q;              % 每一次给 s1 赋初值
        p = 0;               % 用 p 值标记是否搜到 s1 或 s2
        k = 0;               % 用 k 记录站点差
    for j = 1: n
        if ~A(i,j)
            break;
        elseif A(i,j) == s1  % 若搜到 s1,之后在该线路上搜索 s2,并记 p 为 1
            p = p + 1;
            if p == 1
                k = j - k;
                s1 = s2;
            elseif p == 2    % 当 p 值为 2 时,说明已搜到 s2,记录相关信息
                L1 = [L1,i];
                t1 = [t1,3 * k]; % 同时 s1 恢复至原始值,进行下一线路的搜索
                break;
            end
        end
    end
end
```

程序运行后得到结果如下：

```
?[L,t] = DirectLineSearch(242,105,A)
L =
    8
t =
    24
?[L,t] = DirectLineSearch(117,53,A)
L =
    10
t =
    15
?[L,t] = DirectLineSearch(179,201,A)
L =
    7 14
t =
    27
?[L,t] = DirectLineSearch(16,162,A)
L =
    No direct line
t =
    Null
```

在设计算法或循环控制时，应注意信息获取的途径，避免做无用的操作步骤。如果上面这个程序不够优化，它将为后续转车的程序造成不良影响。

附录数据：公交线路信息

线路 1

219-114-88-48-392-29-36-16-312-19-324-20-314-128-76-113-110-213-14-301-115-34-251-95-184-92

线路 2

348-160-223-44-237-147-201-219-321-138-83-161-66-129-254-331-317-303-127-68

线路 3

23-133-213-236-12-168-47-198-12-236-113-212-233-18-127-303-117-231-254-129-366-161-133-181-132

线路 4

201-207-177-144-223-216-48-42-280-140-238-236-158-53-93-64-130-77-264-208-286-123

线路 5

217-272-173-25-33-76-37-27-65-274-234-221-137-306-162-84-325-97-89-24

线路 6

301-82-79-94-41-105-142-118-130-36-252-172-57-20-302-65-32-24-92-218-31

线路 7

184-31-69-179-84-212-99-224-232-157-68-54-201-57-172-22-36-143-218-129-106-101-194

线路 8

57-52-31-242-18-353-33-60-43-41-246-105-28-33-111-77-49-67-27-8-63-39-317-168-12-163

线路 9

217-161-311-25-29-19-171-45-71-173-129-219-210-35-83-43-139-241-78-50

线路 10

136-208-23-117-77-130-68-45-53-51-78-241-139-343-83-333-190-237-251-291-129-173-171-90-42-179-25-311-161-17

线路 11

43-77-111-303-28-65-246-99-54-37-303-53-18-242-195-236-26-40-280-142

线路 12

274-302-151-297-329-123-122-215-218-102-293-86-15-215-186-213-105-128-201-122-12-29-56-79-141-24-74

线路 13

135-74-16-108-58-274-53-59-43-86-85-47-246-108-199-296-261-203-227-146

线路 14

224-22-70-89-219-228-326-179-49-154-251-262-307-294-208-24-201-261-192-264-146-377-172-123-61-235-294-28-94-57-226-18

线路 15

189-170-222-24-92-184-254-215-345-315-301-214-213-210-113-263-12-167-177-313-219-154-349-316-44-52-19

线路 16

233-377-327-97-46-227-203-261-276-199-108-246-227-45-346-243-59-93-274-58-118-116-74-135

事实上，对于编程能力的训练，往往是先从解决一些较为简单的问题入手，然后通过对这些问题修改某些条件、增加难度等不断摸索，在不知不觉中自己的编程能力就已经被提升到了一个新的高度。

2.7 经典案例

计算机程序就是计算机指令的集合，不同的编程语言指令与功能是不一样的。MATLAB 语言是一种面向对象的高级语言，它具有编程效率高、易学易用的优点。本节介绍几种 MATLAB 编程的经典案例。

通过全局变量可以实现 MATLAB 工作区变量空间和多个函数的函数空间共享，这样，多个使用全局变量的函数和 MATLAB 工作区共同维护这一全局变量，任何一处对全局变量的修改，都会直接改变此全局变量的取值。

在应用全局变量时，通常在各个函数内部通过 global variable 语句声明，在命令窗口或脚本 M 文件中也要先通过 global 声明，然后进行赋值。

【例 2-16】 全局变量的使用。

解：在 MATLAB 命令窗口输入：

```
function y = myt(x)
global a;
a = a + 9;
y = cos(x);
```

然后在命令窗口声明全局变量赋值调用：

```
>> global a
>> a = 2
a =
      2
>> myt(pi)
ans =
    - 1
>> cos(pi)
ans =
    - 1
>> a
a =
      11
```

通过上例可见，用 global 将 a 声明为全局变量后，函数内部对 a 的修改也会直接作用到 MATLAB 工作区中，函数调用一次后，a 的值从 2 变为 11。

在 MATLAB 中，函数 polyfit()采用最小二乘法对给定的数据进行多项式拟合，得到该多项式的系数。该函数的调用方式如下：

```
polyfit(x,y,n)
```

找到次数为 n 的多项式系数，对于数据集合{(xi, yi)}，满足差的平方和最小。

```
[p,E] = polyfit(x,y,n)
```

返回同上的多项式 p 和矩阵 E。多项式系数在向量 p 中，矩阵 E 用在 polyval 函数中来计算误差。

【例 2-17】 某数据的横坐标为 $\boldsymbol{x}$=[0.2 0.3 0.5 0.6 0.8 0.9 1.2 1.3 1.5 1.8]，纵坐标为 $\boldsymbol{y}$=[1 2 3 5 6 7 6 5 4 1]，对该数据进行多项式拟合。

解：MATLAB 代码如下：

```
clear all
clc
x = [0.3 0.4 0.7 0.9 1.2 1.9 2.8 3.2 3.7 4.5];
y = [1 2 3 4 5 2 6 9 2 7];
```

```
p5 = polyfit(x,y,5);                    %5 阶多项式拟合
y5 = polyval(p5,x);
p5 = vpa(poly2sym(p5),5)                %显示 5 阶多项式
p9 = polyfit(x,y,9);                    %9 阶多项式拟合
y9 = polyval(p9,x);
figure;                                 %画图显示
plot(x,y,'bo');
hold on;
plot(x,y5,'r:');
plot(x,y9,'g--');
legend('原始数据','5 阶多项式拟合','9 阶多项式拟合');
xlabel('x');
ylabel('y');
```

运行程序后，得到的 5 阶多项式如下：

```
p5 =
    0.8877 * x^5 - 10.3 * x^4 + 42.942 * x^3 - 77.932 * x^2 + 59.833 * x - 11.673
```

运行程序后，得到的输出结果如图 2-9 所示。由图可以看出，使用 5 次多项式拟合时，得到的结果比较差。当采用 9 次多项式拟合时，得到的结果与原始数据符合的比较好。当使用函数 polyfit()进行拟合时，多项式的阶次最大不超过 length(x)－1。

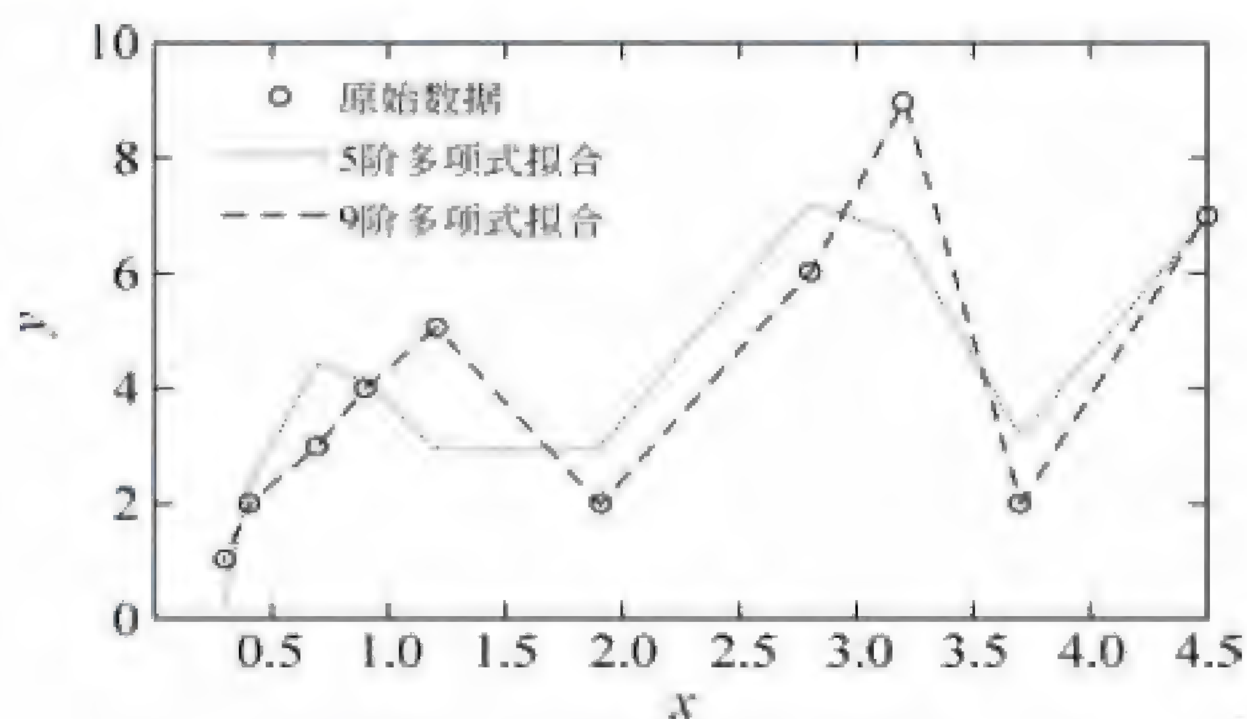


图 2-9　多项式曲线拟合

【例 2-18】 可变长度数据量使用示例。

创建 M 文件，利用 M 文件编辑器，在 M 文件中写入：

```
function varargout = spirallength(d, n, varargin)
%SPIRAL 画出螺旋线或螺旋条带
Nin = length(varargin) + 1;
%error(nargchk(1, Nin, nargin))
if nargout > 1
   error('Too many output arguments!!!')
end
j = sqrt(-1);
phi = 0: pi/20 : n*2*pi;
amp = 0: d/40 : n*d;
```

```
spir = amp .* exp(j*phi);
if nargout == 0
    switch Nin
    case 1
        plot(spir,'b')
    case 2
        d1 = varargin{1};
        amp1 = (0: d/40 : n*d) + d1; spir1 = amp1 .* exp(j*phi);
        plot(spir,'b');hold on;plot(spir1,'b');hold off
        otherwise
        d1 = varargin{1};
        amp1 = (0: d/40 : n*d) + d1; spir1 = amp1 .* exp(j*phi);
        plot(spir,varargin{2:end});hold on;plot(spir1,varargin{2:end});
    end;
    axis('square')
else
    phi0 = 0: pi/1000 : n*2*pi;
    amp0 = 0: d/2000 : n*d;
    spir0 = amp0 .* exp(j*phi0);
    varargout{1} = sum(abs(diff(spir0)));
    if Nin>1
        d1 = varargin{1};
        amp1 = (0: d/2000 : n*d) + d1; spir1 = amp1 .* exp(j*phi);
        varargout{2} = sum(abs(diff(spir1)));
    end;
end
```

在命令行窗口输入：

```
subplot(1,3,1), spirallength(2,2,1)
subplot(1,3,2), spirallength(2,2,1,'Marker','o')
subplot(1,3,3), spirallength(2,2,1,'r--', 'LineWidth',2)
```

输出结果如图 2-10 所示。

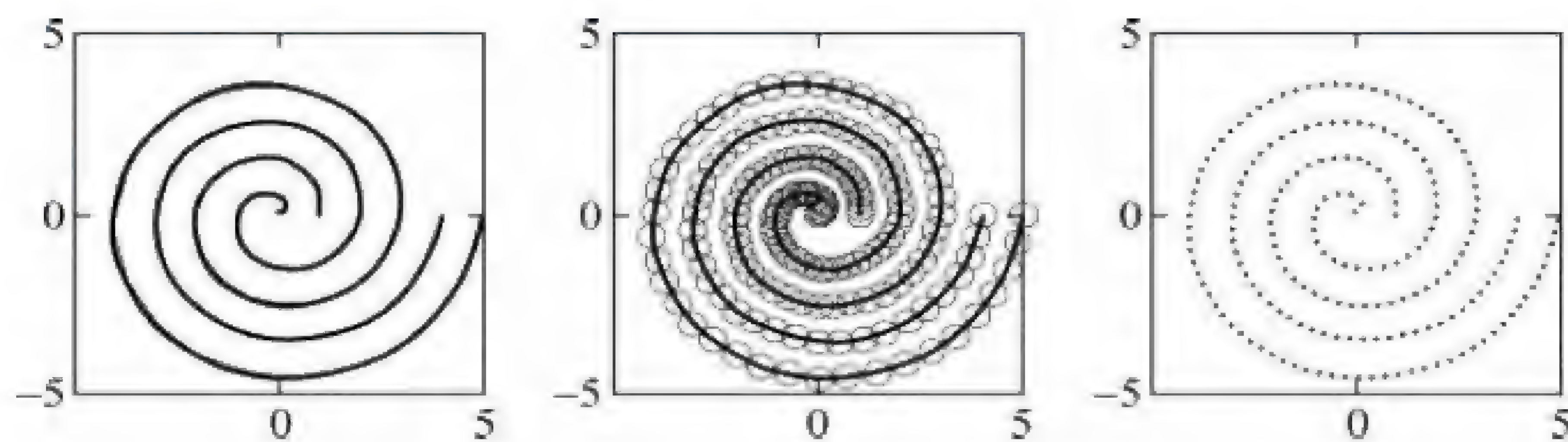


图 2-10　spirallength. m 运行结果

【例 2-19】 完整的 M 文件示范。

解：创建 M 文件，利用 M 文件编辑器，在 M 文件中写入：

```
function spir_len = spirallength(d, n, lcolor)
% CIRCLE plot a circle of radius as r in the provided color and calculate its area
% d: 螺旋的旋距
%   n: 螺旋的圈数
%   lcolor: 画图线的颜色
%   spir_len: 螺旋的周长
% spirallength(d, n): 利用蓝色以预设参数的螺旋线
% spirallength(d, n,lcolor): 利用 lcolor 颜色以预设参数的螺旋线
% spir_len = spirallength(d, n): 计算螺旋线的周长,并用蓝色填充螺旋线
% spir_len = spirallength(d, n,lcolor): 计算螺旋线的周长,并用 lcolor 颜色填充螺旋线
% 编写于 2009.7.6,修改于 2011.8.8       程序员: 01
if nargin > 3
    error('输入变量过多!');
elseif nargin == 2
    lcolor = 'b';
end
j = sqrt(-1);
phi = 0: pi/1000 : n * 2 * pi;
amp = 0: d/2000 : n * d;
spir = amp . * exp(j * phi);
if nargout == 1
    spir_len = sum(abs(diff(spir)));
    fill(real(spir), img(spir), lcolor)
elseif nargout == 0
    plot(spir,lcolor)
else
    error('输出变量过多!');
end
axis('square')
```

在命令行窗口输入:

```
spirallength(0.25, 4, 'k:')
```

输出结果如图 2-11 所示。

图论算法在计算机科学中扮演着很重要的角色,它提供了对很多问题都有效的一种简单而系统的建模方式。图论中的图是由若干给定的点及连接两点的线所构成的图形,这种图形通常用来描述某些事物之间的某种特定关系,用点代表事物,用连接两点的线表示相应两个事物间具有的关系。

【例 2-20】 很多问题都可以转化为图论问题,然后用图论的基本算法加以解决。利用 MATLAB 编程,实现图论算法中的最小生成树、最短路的 Dijkstra 算法、Ford 最短路算法、层次分析、灰色关联性分析和灰色预测。

解:(1) 最小生成树 MATLAB 编程如下:

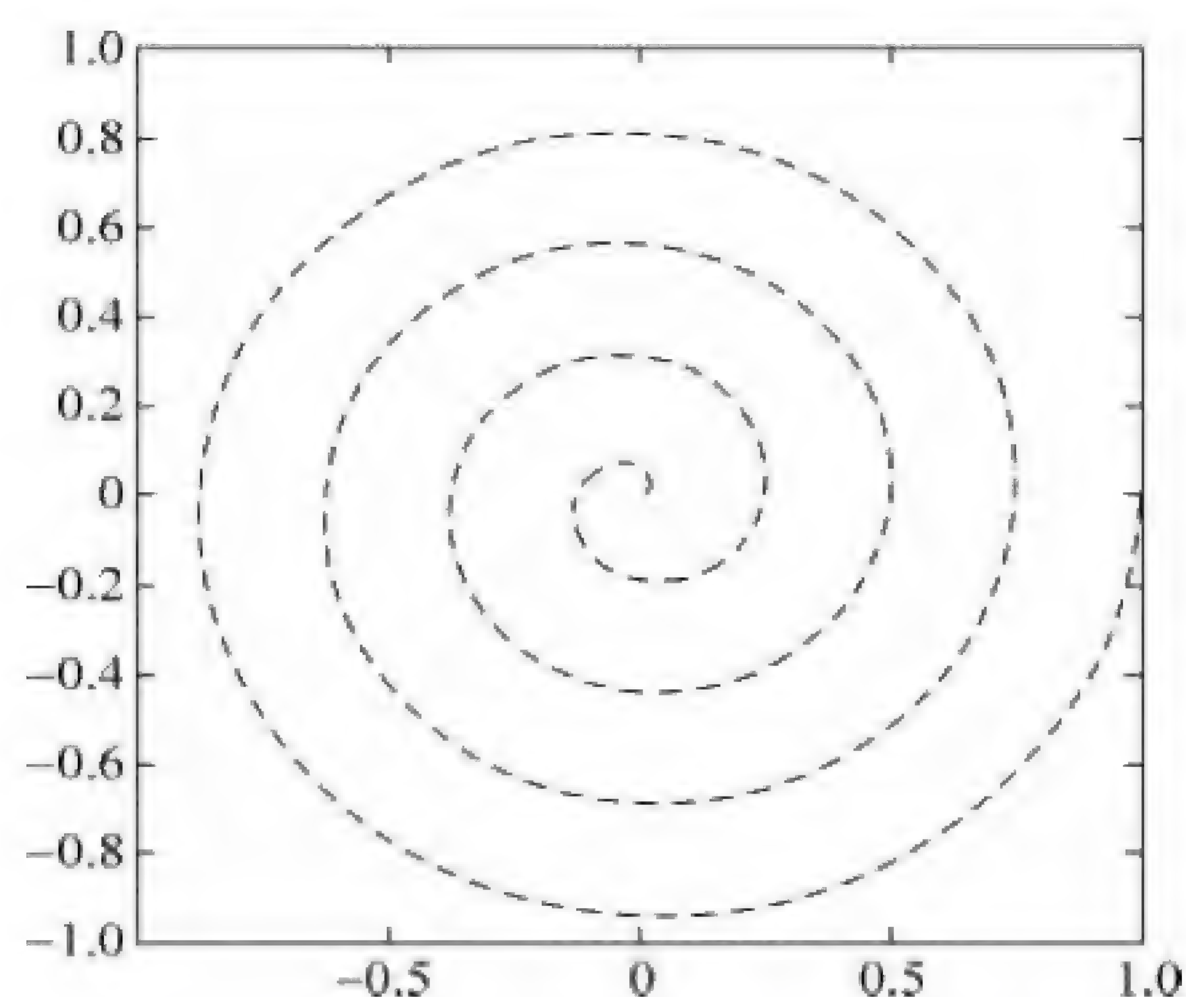


图 2-11 spirallength.m 运行结果图

```
function [w,E] = MinTree(A)
%避圈法求最小生成树
%A 为图的赋权邻接矩阵
%w 记录最小树的权值之和,E 记录最小树上的边
n = size(A,1);
for i = 1: n
A(i,i) = inf;
end
s1 = [];s2 = [];                %s1,s2 记录一条边上的两个顶点
w = 0; k = 1; %k 记录顶点数
T = A + inf;
T(1,: ) = A(1,: );
A(: ,1) = inf;
while k < n
    [p1,q1] = min(T);           %q1 记录行下标
    [p2,q2] = min(p1);
    i = q1(q2);
    s1 = [s1,i];s2 = [s2,q2];
    w = w + p; k = k + 1;
    A(: ,q2) = inf;             %若此顶点已被连接,则切断此顶点的入口
    T(q2,: ) = A(q2,: );        %在 T 中并入此顶点的出口
    T(: ,q2) = inf;
end
E = [s1;s2];                    %E 记录最小树上的边
```

(2)最短路的 Dijkstra 算法 MATLAB 编程如下:

```
function [d,path] = ShortPath(A,s,t)
%Dijkstra 最短路算法实现,A 为图的赋权邻接矩阵
%当输入参数含有 s 和 t 时,求 s 到 t 的最短路
%当输入参数只有 s 时,求 s 到其他顶点的最短路
%返回值 d 为最短路权值,path 为最短路径
```

```
if nargin == 2
    flag = 0;
elseif nargin == 3
    flag = 1;
end
n = length(A);
for i = 1: n
    A(i,i) = inf;
end
V = zeros(1,n);                     % 存储 lamda(由来边)标号值
D = zeros(1,n);                     % 用 D 记录权值
T = A + inf;                        % T 为标号矩阵
T(s,: ) = A(s,: );                  % 先给起点标号
A(: ,s) = inf;                      % 关闭进入起点的边
for k = 1: n - 1
    [p,q] = min(T);                 % p 记录各列最小值,q 为对应的行下标
    q1 = q;                         % 用 q1 保留行下标
    [p,q] = min(p);                 % 求最小权值及其列下标
    V(q) = q1(q);                   % 求该顶点 lamda 值
    if flag&q == t
        d = p;                      % 求最短路权值
        break;
    else                            % 修改 T 标号
        D(q) = p;                   % 求最短路权值
        A(: ,q) = inf;              % 将 A 中第 q 列的值改为 inf
        T(q,: ) = A(q,: ) + p;      % 同时修改从顶点 q 出去的边上的权值
        T(: ,q) = inf;              % 顶点 q 点已完成标号,将进入 q 的边关闭
    end
end
if flag                             % 输入参数含有 s 和 t,求 s 到 t 的最短路
    path = t;                       % 逆向搜索路径
    while path(1)~ = s
        path = [V(t),path];
        t = V(t);
    end
else                                % 输入参数只有 s,求 s 到其他顶点的最短路
    for i = 1: n
        if i~ = s
            path0 = i;v0 = i;       % 逆向搜索路径
            while path0(1)~ = s
                path0 = [V(i),path0];
                i = V(i);
            end
            d = D; path(v0) = {path0}; % 将路径信息存放在元胞数组中
                                    % 在命令行窗口显示权值和路径
            disp([int2str(s),' ->',int2str(v0),' d = ',...
            int2str(D(v0)),' path =  ',int2str(path0)]);
        end
    end
end
```

（3）Ford 最短路算法 MATLAB 编程如下：

该算法用于求解一个赋权图中 sv 到的最短路，并且对于权值的情况同样适用。

```
function [w,v] = Ford(W,s,t)
%W 为图的带权邻接矩阵,s 为发点,t 为终点
%返回值 w 为最短路的权值之和,v 为最短路线上的顶点下标
n = length(W);
d(: ,1) = (W(s,: ))';           %求 d(vs,vj) = min{d(vs,vi) + wij} 的解, 用 d 存放
                                %d(t)(v1,vj), 赋初值为 W 的第 s 行,以列存放
j = 1;
while j
    for i = 1: n
        b(i) = min(W(: ,i) + d(: ,j));
    end
    j = j + 1;
    d = [d,b'];
    if d(: ,j) == d(: ,j - 1)   %若找到最短路, 跳出循环
        break ;
    end
end
w = d(t,j);                     %记录最短路的权值之和
v = t;                          %用数组 v 存放最短路上的顶点,终点为 t
while v(1)~ = s
    for i = n: - 1:1
        if i~ = t&W(i,t) + d(i,j) == d(t,j)
            break;
        end
    end
    v = [i,v];
    t = i;
end
```

（4）层次分析 MATLAB 编程如下：

在层次分析中，该算法用于根据成对比较矩阵求近似特征向量。

```
function [w,lam,CR] = ccfx(A)
%A 为成对比较矩阵,返回值 w 为近似特征向量,
%lam 为近似最大特征值 maxλ,CR 为一致性比率
n = length(A(: ,1));
a = sum(A);
B = A;                          %用 B 代替 A 做计算
for j = 1: n                    %将 A 的列向量归一化
    B(: ,j) = B(: ,j)./a(j);
end
s = B(: ,1);
for j = 2: n
    s = s + B(: ,j);
end
```

```
c = sum(s);                        % 和法计算近似最大特征值 maxλ
w = s./c;
d = A * w;
lam = 1/n * sum((d./w));
CI = (lam - n)/(n - 1);            % 一致性指标
RI = [0,0,0.58,0.90,1.12,1.24,1.32,1.41,1.45,1.49,1.51];
%RI 为随机一致性指标
CR = CI/RI(n);                     % 求一致性比率
if CR > 0.1
    disp('没有通过一致性检验');
    else disp('通过一致性检验');
end
```

(5) 灰色关联性分析 MATLAB 编程如下：

当系统的行为特征只有一个因子 x0，该算法用于求解各种因素 xi 对 x0 的影响大小。

```
function s = Glfx(x0,x)           % x0(行向量)为因子,x 为因素集
[m,n] = size(x);
B = [x0;x];
k = m + 1;                        % k 为 B 的行数
c = B(:,1);                       % 对序列进行无量纲化处理
for j = 1:n
    B(:,j) = B(:,j)./c;
end
for i = 2:k                       % 求参考序列对各比较序列的绝对差
    B(i,:) = abs(B(i,:) - B(1,:));
end
A = B(2:k,:);                     % 求关联系数
a = min(min(A));
b = max(max(A));
for i = 1:m
    for j = 1:n
        r1(i,j) = r1(i,j) * (a + 0.5 * b)/(A(i,j) + 0.5 * b);
    end
end
s = 1/n * (r1 * ones(m,1));       % 比较序列对参考序列 x0 的灰关联度
```

(6) 灰色预测 MATLAB 编程如下：

该算法用灰色模型中的 GM(1,1)模型做预测。

```
function [s,t] = huiseyc(x,m)
%x 为待预测变量的原值,为其预测 m 个值
[m1,n] = size(x);
if m1~ = 1             % 若 x 为列向量,则将其变为行向量放入 x0
    x0 = x';
else
```

```
    x0 = x;
end
n = length(x0);
c = min(x0);
if c < 0                         % 若 x0 中有小于 0 的数,则作平移,使每个数字都大于 0
    x0 = x0 - c + 1;
end
x1 = (cumsum(x0))';              % x1 为 x0 的 1 次累加生成序列,即 AGO
for k = 2: n
    r(k - 1) = x0(k)/x1(k - 1);
end
rho = r,                         % 光滑性检验
for k = 2: n
    z1(k - 1) = 0.5 * x1(k) + 0.5 * x1(k - 1);
end
B = [ - z1',ones(n - 1,1)];
YN = (x0(2: n))';
a = (inv(B' * B)) * B' * YN;
y1(1) = x0(1);
for k = 2: n + m                 % 预测 m 个值
    y1(k) = (x0(1) - a(2)/a(1)) * exp( - a(1) * (k - 1)) + a(2)/a(1);
end
y(1) = y1(1);
for k = 2: n + m
    y(k) = y1(k) - y1(k - 1);    % 还原
end
if c < 0
    y = y + c - 1;
end
y;
e1 = x0 - y(1: n);
e = e1(2: n),                    % e 为残差
for k = 2: n
    dd(k - 1) = abs(e(k - 1))/x0(k);
end
dd;
d = 1/(n - 1) * sum(dd);
f = 1/(n - 1) * abs(sum(e));
s = y;
t = e;
```

本章小结

本章首先简单介绍了 MATLAB 编程概述和编程原则，其次详细讲述了分支结构、循环结构以及其他控制程序指令，并通过案例说明如何用 MATLAB 进行程序设计，如何编写清楚、高效的程序，最后对 MATLAB 程序调试做了简单介绍，并指出了一些使用技巧和编程者常犯的错误。

第3章 MATLAB绘图

强大的绘图功能是 MATLAB 的特点之一，MATLAB 提供了一系列的绘图函数，用户不需要过多的考虑绘图的细节，只需要给出一些基本参数就能得到所需图形。此外，MATLAB 还对绘出的图形提供了各种修饰方法，使图形更加美观、精确。

学习目标：

(1) 了解 MATLAB 数据绘图；

(2) 熟练掌握 MATLAB 中二维绘图；

(3) 熟练掌握 MATLAB 中三维绘图；

(4) 了解 MATLAB 多种特殊图形。

3.1 数据图像绘制简介

数据可视化的目的在于：通过图形，从一堆杂乱的离散数据中观察数据间的内在关系，感受由图形所传递的内在本质。

MATLAB 一向注重数据的图形表示，并不断地采用新技术改进和完备其可视化功能。

3.1.1 离散数据可视化

任何二元实数标量对(x_a,y_a)可以在平面上表示一个点；任何二元实数向量对(X,Y)可以在平面上表示一组点。

对于离散实函数 $y_n=f(x_n)$，当 $X=[x_1,x_2,\cdots,x_n]$以递增或递减的次序取值时，有 $Y=[y_1,y_2,\cdots,y_n]$。这样，该向量对用直角坐标序列点图示时，实现了离散数据的可视化。

在科学研究中，当处理离散量时，可以用离散序列图来表示离散量的变化情况。MATLAB 用 stem 命令来实现离散图形的绘制，stem 命令有以下几种：

1. stem(y)

以 $x=1,2,3\cdots$作为各个数据点的 x 坐标，以向量 y 的值为 y 坐

标，在(x,y)坐标点画一个空心小圆圈，并连接一条线段到x轴。

【例 3-1】 用 stem 函数绘制一个离散序列图。

解： 依据题意编写 MATLAB 代码如下：

```
clear all
clc
figure(1)
X = linspace(0,2 * pi,25)';
Y = (cos(2 * X));
stem(X,Y,'LineStyle','-.',...
     'MarkerFaceColor','red',...
     'MarkerEdgeColor','green')
```

输出图形如图 3-1 所示。

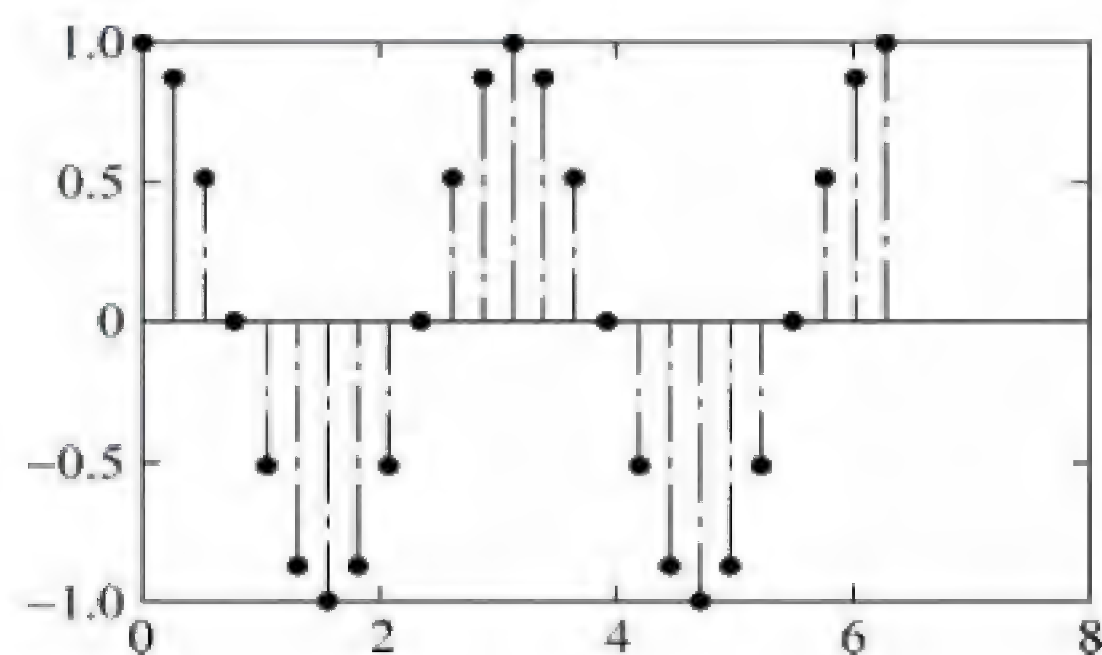


图 3-1 绘制的离散序列图

2. stem(x,y,'option')

以x向量的各个元素为x坐标，以y向量的各个对应元素为y坐标，在(x,y)坐标点画一个空心小圆圈，并连接一条线段到x轴。option 选项表示绘图时的线型、颜色等设置。

3. stem(x,y,'filled')

以x向量的各个元素为x坐标，以y向量的各个对应元素为y坐标，在(x,y)坐标点画一个空心小圆圈，并连接一条线段到x轴。

【例 3-2】 用 stem 函数绘制一个线型为圆圈的离散序列图。

解： 依据题意编写 MATLAB 代码如下：

```
clear all
clc
figure(1)
x = 0:25;
y = [exp(-.04 * x).* cos(x);exp(.04 * x).* cos(x)]';
h = stem(x,y);
set(h(1),'MarkerFaceColor','blue')
set(h(2),'MarkerFaceColor','red','Marker','square')
```

输出图形如图 3-2 所示。

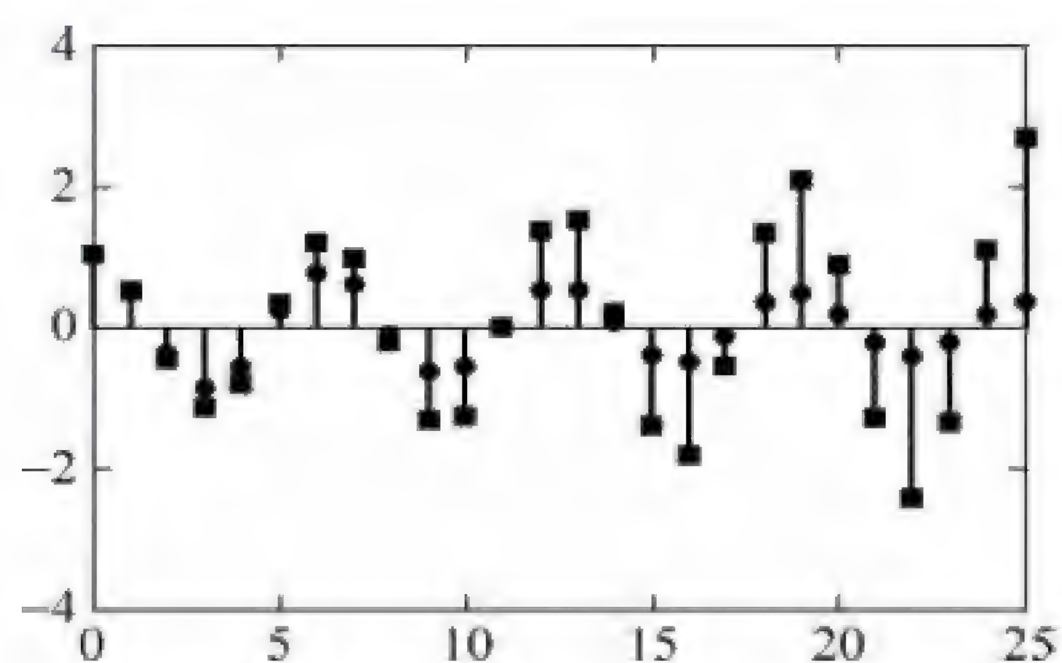


图 3-2 绘制的离散序列图

除了可以使用 stem 命令之外，使用离散数据也可以画离散图形。

【例 3-3】 用图形表示离散函数。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
n = 0:10;                    % 产生一组 10 个自变量函数 Xn
y = 1./abs(n - 6);           % 计算相应点的函数值 Yn
plot(n, y, 'r * ', 'MarkerSize', 25)
                             % 用尺寸 15 的红星号标出函数点
grid on                      % 画出坐标方格
```

输出图形如图 3-3 所示。

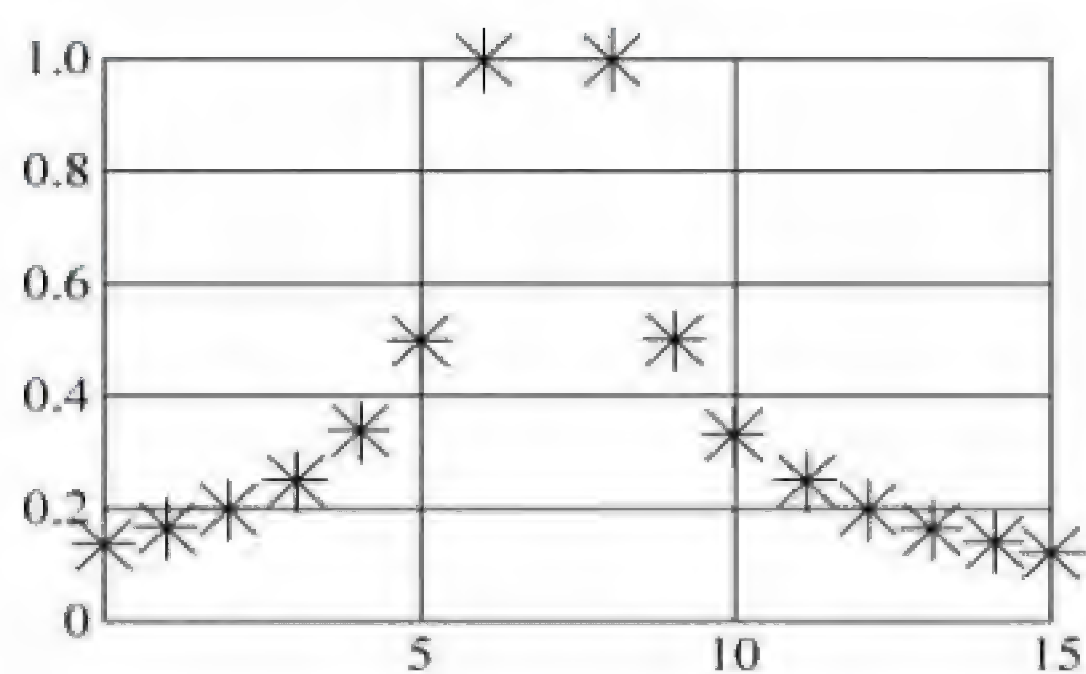


图 3-3 绘制的离散函数图形

【例 3-4】 画出函数 $y = \mathrm{e}^{-at}\sin bt$ 的茎图。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
a = 0.03;
b = 0.8;
t = 0:1:60;
y = exp( - a * t). * sin(b * t) ;
plot(t,y)
title('茎图')
```

运行后得到的输出图形如图 3-4 所示。

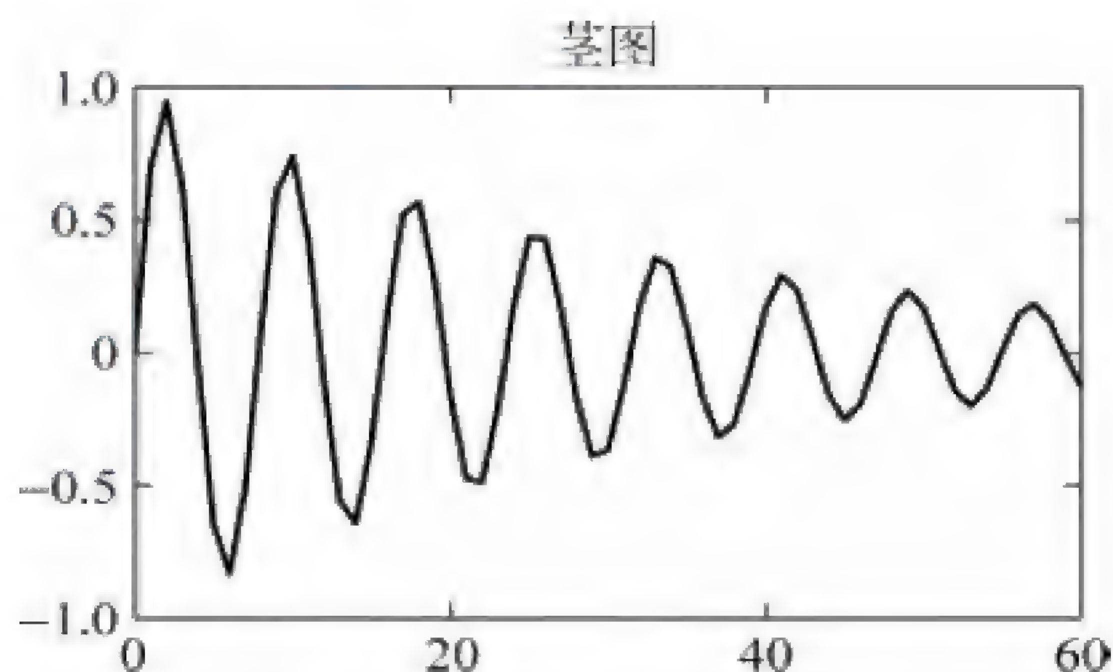


图 3-4　绘制的连续图形

二维的茎图函数为 stem(t,y)，具体代码如下：

```
clear all
clc
a=0.03;
b=0.8;
t = 0:1:60;
y=exp(-a*t).*sin(b*t) ;
stem(t,y)
xlabel('Time')
ylabel('stem')
title('茎图')
```

输出二维的茎图如图 3-5 所示。

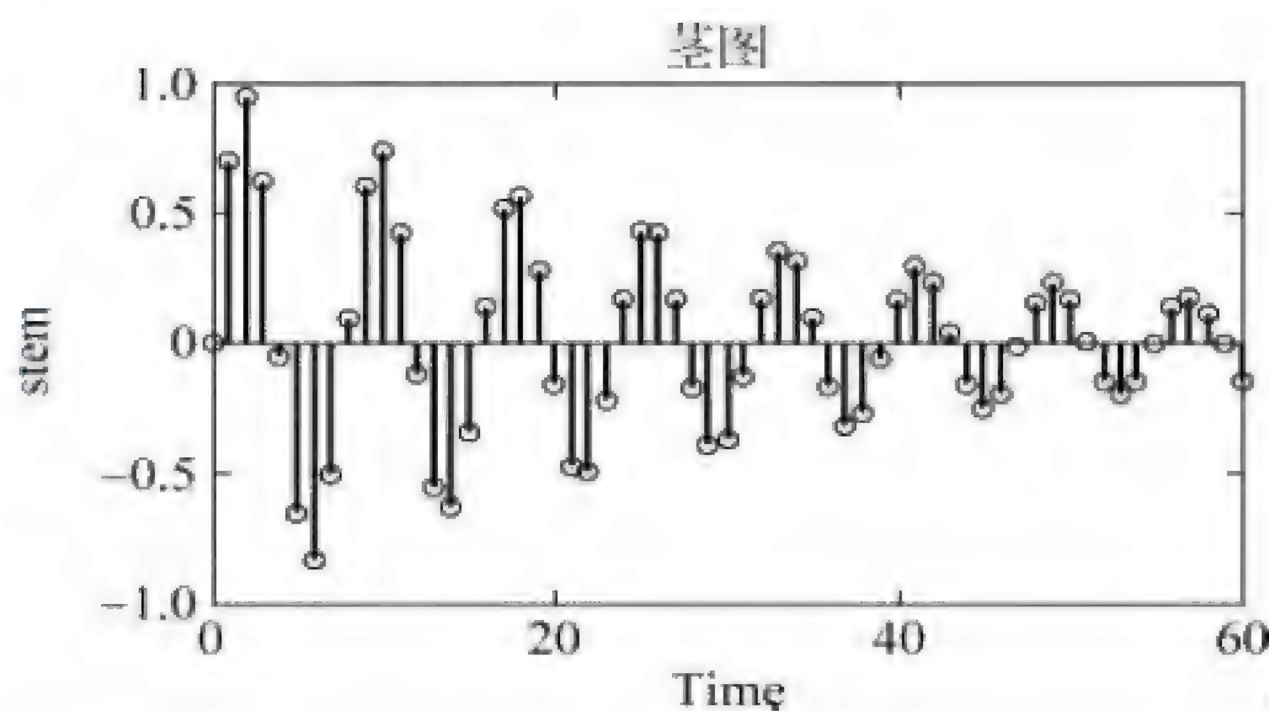


图 3-5　二维的茎图

3.1.2　连续函数可视化

对于连续函数可以取一组离散自变量，然后计算函数值，与离散数据的显示方式一样。

一般画函数或方程式的图形，都是先标注几个图形上的点，进而再将点连接即为函数图形，其点愈多图形越平滑。MATLAB 在简易二维画图中也是相同做法，必须先点出 x 和 y 坐标（离散数据），再将这些点连接，语法如下：

```
plot(x,y)   %x为图形上x坐标向量,y为其对应的y坐标向量
```

【例 3-5】 用图形表示连续调制波形 $y=\sin(t)\sin(7t)$。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
t1 = (0:13)/13 * pi;                  % 自变量取13个点
y1 = sin(t1). * sin(7 * t1);          % 计算函数值
t2 = (0:40)/40 * pi;                  % 自变量取51个点
y2 = sin(t2). * sin(7 * t2);
subplot(2,2,1);                       % 在子图3上画图
plot(t1,y1,'r.');                     % 用红色的点显示
axis([0,pi,-1,1]);                    % 定义坐标大小
title('子图3');                        % 显示子图标题
%子图2用红色的点显示
subplot(2,2,2);
plot(t2,y2,'r.');
axis([0,pi,-1,1]);
title('子图2')
%子图3用直线连接数据点和红色的点显示
subplot(2,2,3);
plot(t1,y1,t1,y1,'r.')
axis([0,pi,-1,1]);
title('子图3')
%子图4用直线连接数据点
subplot(2,2,4);
plot(t2,y2);
axis([0,pi,-1,1]);
title('子图4')
```

输出图形如图 3-6 所示。

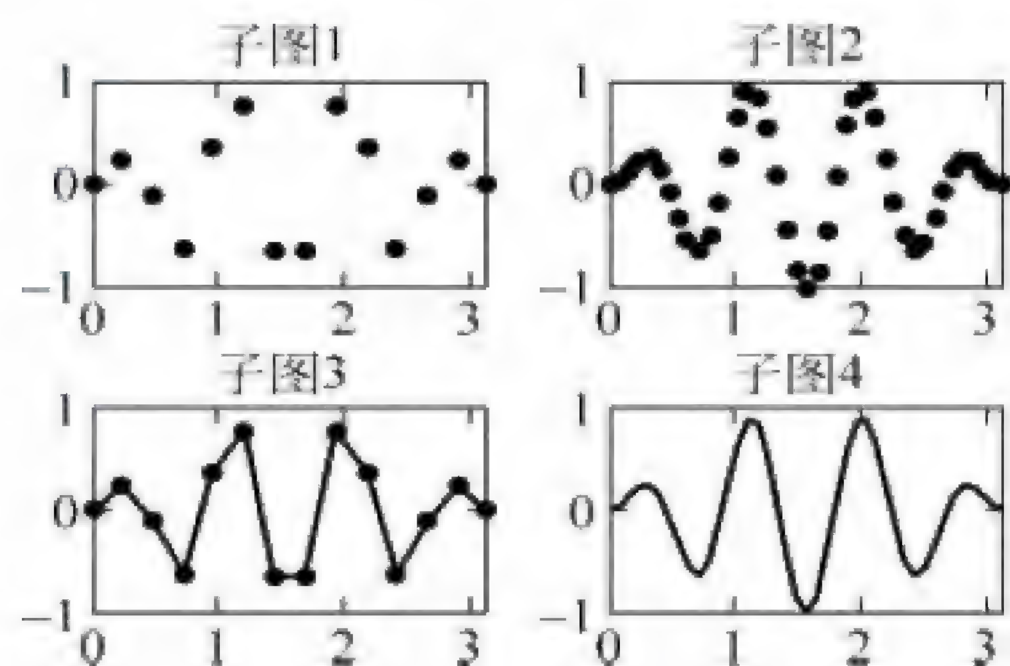


图 3-6 输出图形

【例 3-6】 分别取 8、40、80 个点，绘制 $y=2\sin(x)$，$x\in[0,2\pi]$的图形。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
x8 =  linspace(0,2 * pi,8);                    % 在 0 到 2π 间,等分取 8 个点
y8  = 2 *  sin(x8);                            % 计算 x 的正弦函数值
plot(x8,y8);                                   % 进行二维平面描点作图
```

输出 8 个点图形如图 3-7 所示。

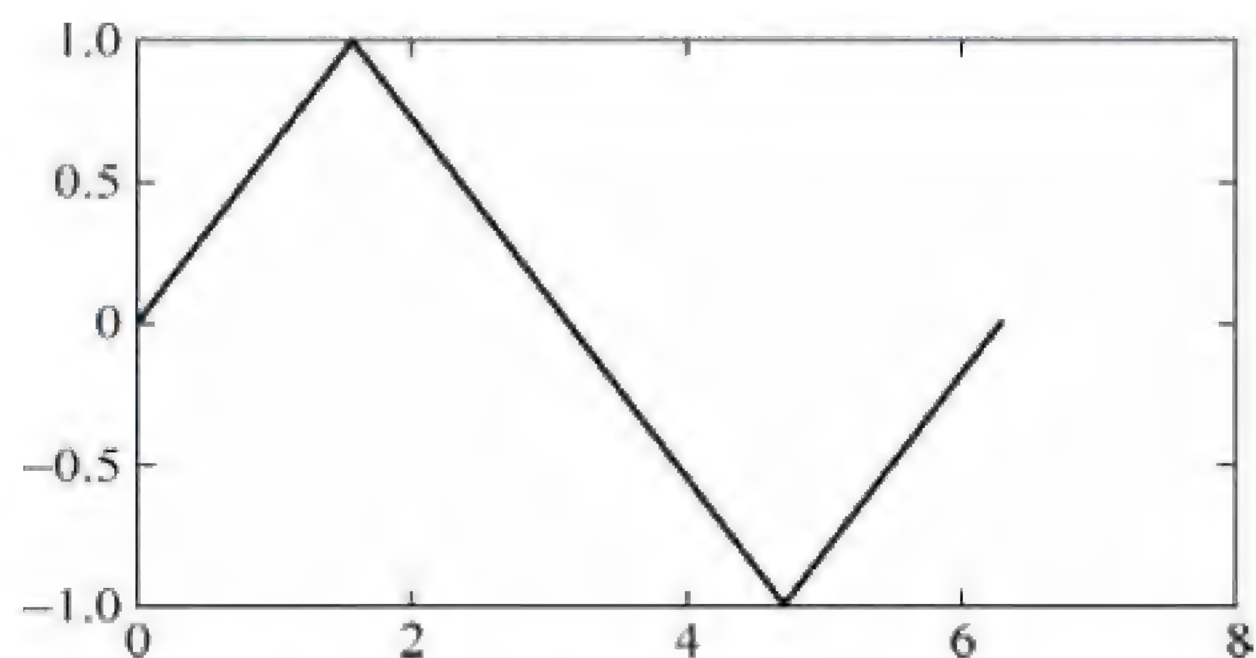


图 3-7　绘制 8 个点函数波形

依据题意编写 MATLAB 代码如下：

```
clear all
clc
x40 =  linspace(0,2 * pi,40);                 % 在 0 到 2π 间,等分取 40 个点
y40  = 2 *  sin(x40);                         % 计算 x 的正弦函数值
plot(x40,y40);                                % 进行二维平面描点作图
title('40 个点绘图')
```

输出 40 个点图形如图 3-8 所示。

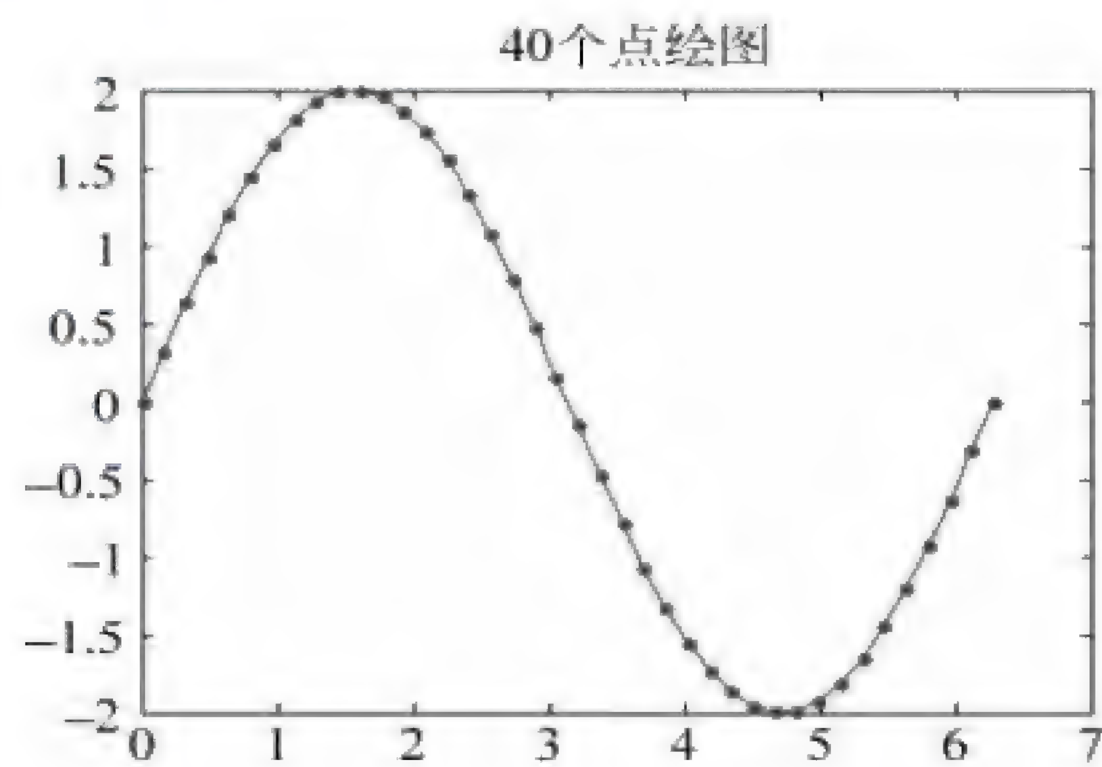


图 3-8　绘制 40 个点函数波形

依据题意编写 MATLAB 代码如下：

```
clear all
clc
x80  = linspace(0,2 * pi,80);                  % 在 0 到 2π 间,等分取 80 个点
y80  =  2 * sin(x80);                          % 计算 x 的正弦函数值
plot(x80,y80);                                 % 进行二维平面描点作图
title('80 个点绘图')
```

输出 80 个点图形如图 3-9 所示。

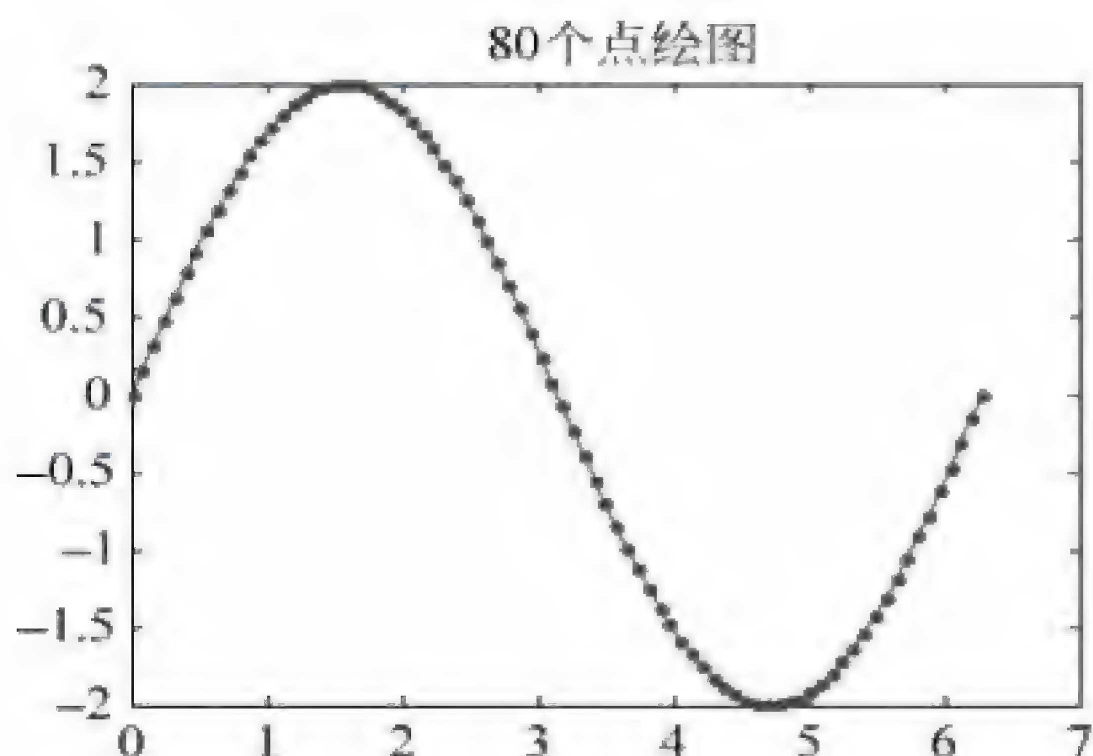


图 3-9　绘制 80 个点函数波形

3.2　二维绘图

MATLAB 不但擅长与矩阵相关的数值运算，而且还提供了许多在二维和三维空间内显示可视信息的函数，利用这些函数可以绘制出所需的图形。本章重点介绍二维绘图的基础内容。

3.2.1　二维图形基本绘图命令 plot

二维图形绘图命令 plot 调用格式如下：

1. plot(X,'s')

X 是实向量时，以向量元素的下标为横坐标，元素值为纵坐标画一连续曲线；X 是实矩阵时，按列绘制每列元素值对应其下标的曲线，曲线数目等于 X 矩阵的列数；X 是复数矩阵时，按列分别以元素实部和虚部为横、纵坐标绘制多条曲线。

2. plot(X,Y,'s')

X、Y 是同维向量时，则绘制以 X、Y 元素为横、纵坐标的曲线；X 是向量，Y 是有一维与 X 等维的矩阵时，则绘出多根不同彩色的曲线，曲线数目等于 Y 的另一维数，X 作为这些曲线的共同坐标；X 是矩阵，Y 是向量时，情况与上相同，Y 作为共同坐标；X、Y 是同维实矩阵时，则以 X、Y 对应的元素为横、纵坐标分别绘制曲线，曲线数目等于矩阵的列数。

3. plot(X1,Y1,'s1',X2,Y2,'s2',…)

s、s1、s2 用来指定线型、色彩、数据点形的字符串。

【例 3-7】 绘制一组幅值不同的正弦函数。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
t = (0:pi/8:2 * pi)';          % 横坐标列向量
k = 0.2:0.1:1;                 % 9 个幅值
Y = sin(t) * k;                % 9 条函数值矩阵
plot(t,Y)
title('函数值曲线')
```

得到图形如图 3-10 所示。

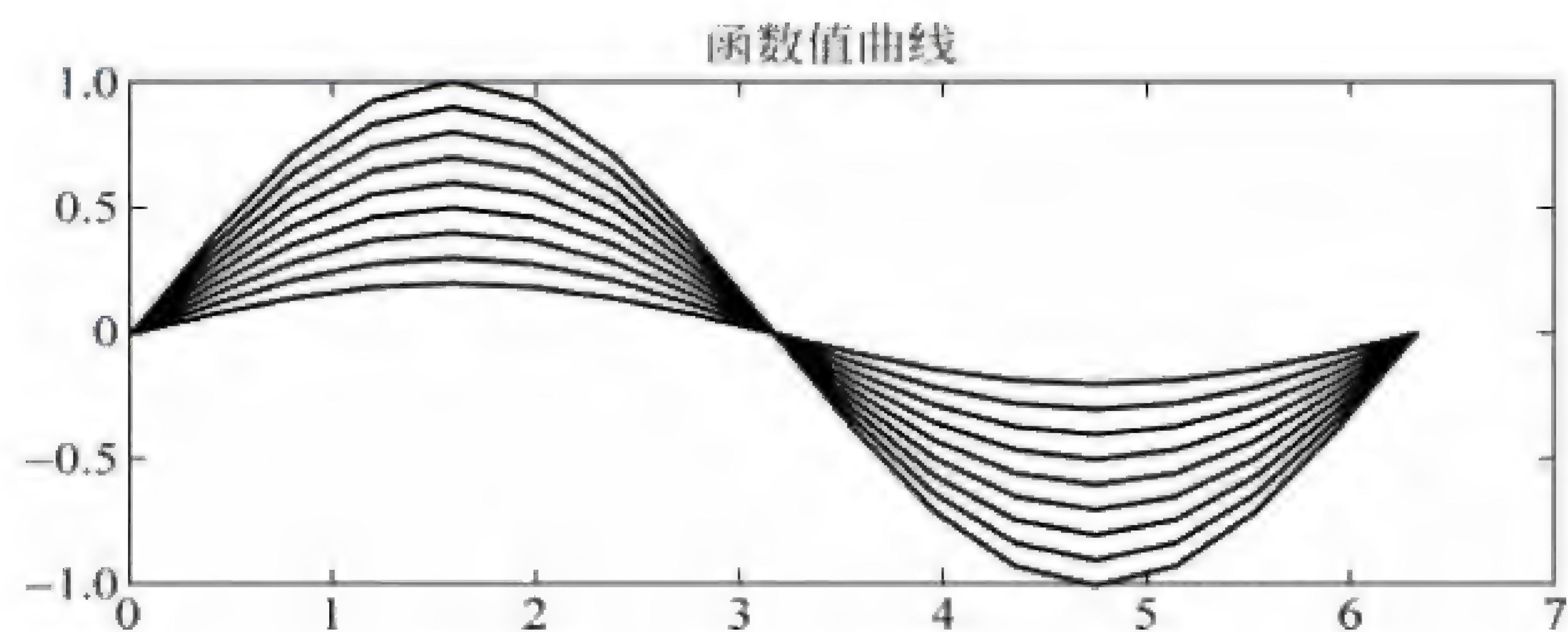


图 3-10 幅值不同的余弦函数

【例 3-8】 用图形表示连续调制波形及其包络线。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
t = (0:pi/100:3 * pi)';
y1 = sin(t) * [1, - 1];
y2 = sin(t). * sin(7 * t);
t3 = pi * (0:7)/7;
y3 = sin(t3). * sin(7 * t3);
plot(t,y1,'r:',t,y2,'b',t3,y3,'b * ')
axis([0,2 * pi, - 1,1])
title('连续调制波形及其包络线')
```

得到图形如图 3-11 所示。

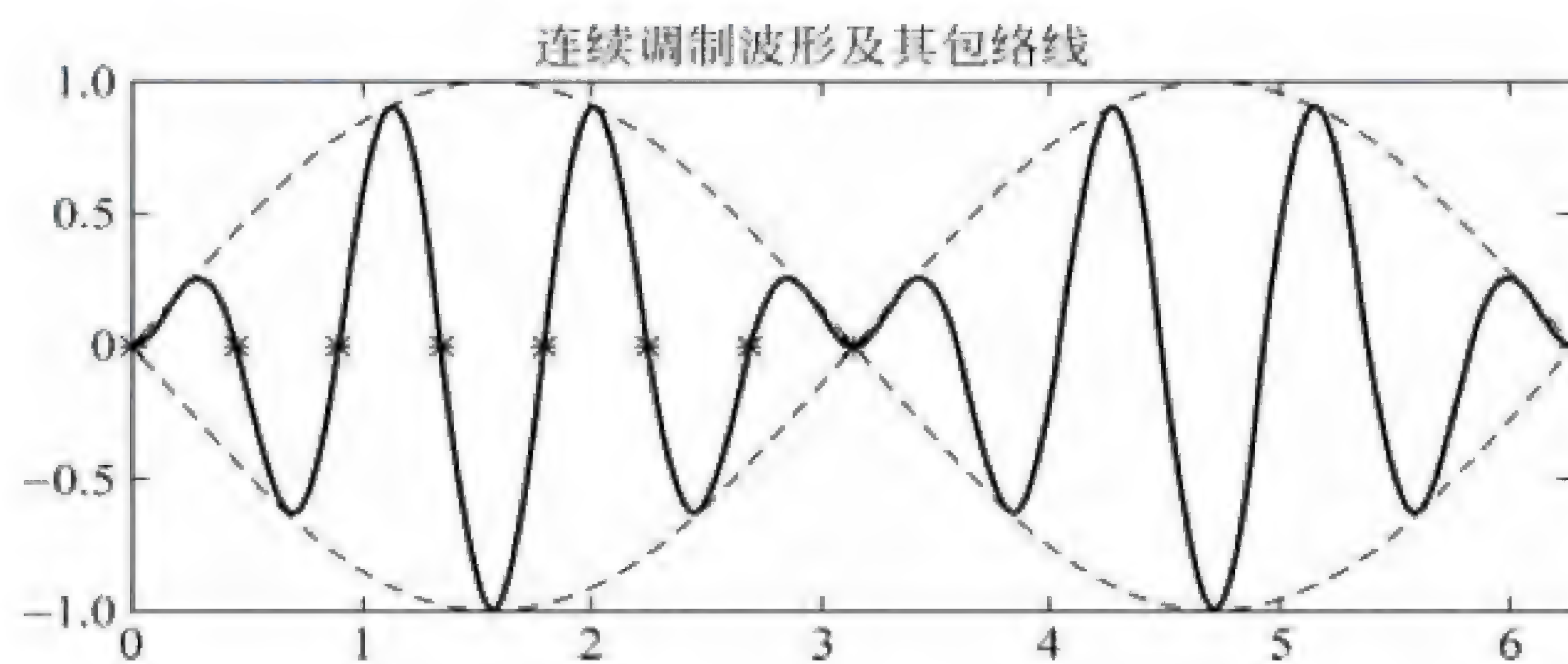


图 3-11 连续调制波形及其包络线

【例 3-9】 用复数矩阵形式画图形。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
t = linspace(0,2 * pi,100)';
X = [cos(t),cos(2 * t),cos(3 * t)] + i * sin(t) * [1,1,1];
plot(X),axis square;
legend('1','2','3')
```

得到图形如图 3-12 所示。

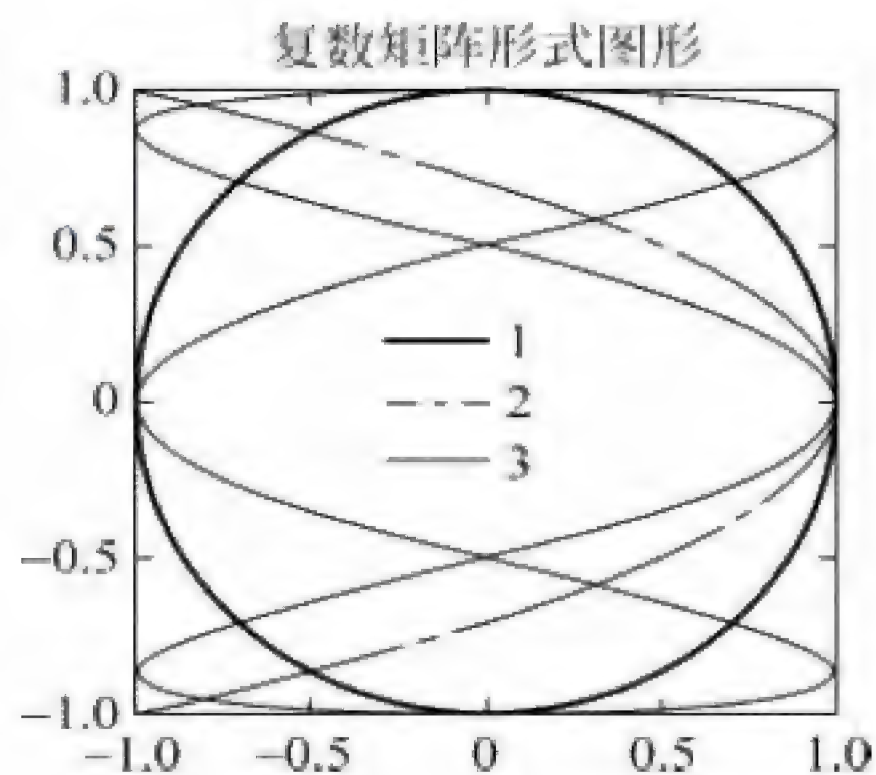


图 3-12 用复数矩阵形式画的图形

【例 3-10】 采用模型$\frac{x^2}{a^2}+\frac{y^2}{23-a^2}=1$画一组椭圆。

解：依据题意编写 MATLAB 代码如下：

```
clear all
clc
th = [0:pi/50:2 * pi]';
a  = [0.5:.5:4.5];
X  = cos(th) * a;
Y  = sin(th) * sqrt(23 - a.^2);
plot(X,Y)
axis('equal')
xlabel('x')
ylabel('y')
title('椭圆图形')
```

得到如图 3-13 所示椭圆图形。

3.2.2 二维图形的修饰

MATLAB 在绘制二维图形的时候，还提供了多种修饰图形的方法，包括色彩、线型、点型、坐标轴等方面。本节详细介绍 MATLAB 中常见的二维图形修饰方法。

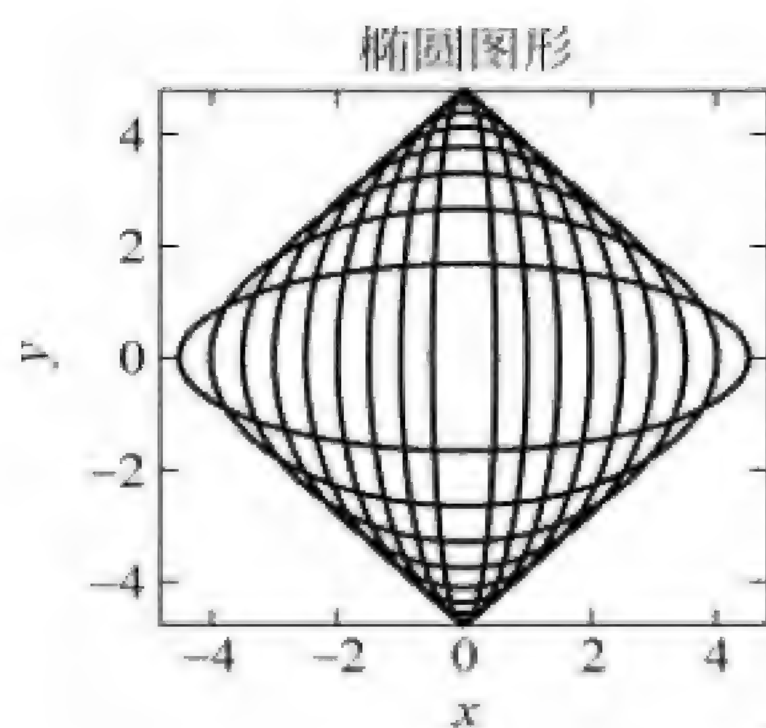


图 3-13　椭圆图形

1. 坐标轴的调整

在一般情况下不必选择坐标系，MATLAB 可以自动根据曲线数据的范围选择合适的坐标系，从而使曲线尽可能清晰地显示出来。但是，如果对 MATLAB 自动产生的坐标轴不满意，可以利用 axis 命令对坐标轴进行调整。

```
axis(xmin,xmax,ymin,ymax)
```

这个命令将所画图形的 x 轴的大小范围限定在 xmin 和 xmax 之间，y 轴的大小范围限定在 ymin 和 ymax 之间。

在 MATLAB 中，坐标轴控制的方法见表 3-1 所示。

表 3-1　坐标轴控制方法

坐标轴控制方式、取向和范围		坐标轴的高宽比	
axis auto	使用默认设置	axis epual	纵、横轴采用等长刻度
axis manual	使用当前坐标范围不变	axis fill	Manual 方式起作用，坐标充满整个绘图区
axis off	取消轴背景	axis image	同 epual 且坐标紧贴数据范围
axis on	使用轴背景	axis normal	默认矩形坐标系
axis ij	矩阵式坐标，原点在左上方	axis square	产生正方形坐标系
axis xy	直角坐标，原点在左下方	axis tight	数据范围设为坐标范围
axis(V);V = [x1, x2, y1, y2]; V = [x1,x2, y1, y2, z1, z2]	人工设定坐标范围	axis vis3d	保持高、宽比不变，用于三维旋转时避免图形大小变化

【例 3-11】 尝试使用不同的 MATLAB 坐标轴控制指令，观察各种坐标轴控制指令的影响。

解：依据题意编写代码如下：

```
clear all
clc
t=0:2*pi/97:2*pi;
x=1.13*cos(t);
y=3.23*sin(t);                    %椭圆
```

```
subplot(2,3,1),
plot(x,y),
grid on;                    % 子图 3
axis normal,
title('normal');
subplot(2,3,2),
plot(x,y),
grid on;                    % 子图 2
axis equal,
title('equal')
subplot(2,3,3)
plot(x,y)
grid on;                    % 子图 3
axis square
title('Square')
subplot(2,3,4)
plot(x,y),
grid on                     % 子图 4
axis image
box off
title('Image and Box off')
subplot(2,3,5),
plot(x,y)
grid on                     % 子图 5
axis image fill
box off
title('Image Fill')
subplot(2,3,6)
plot(x,y)
grid on;                    % 子图 3
axis tight,
box off,
title('Tight')
```

得到图形如图 3-14 所示。

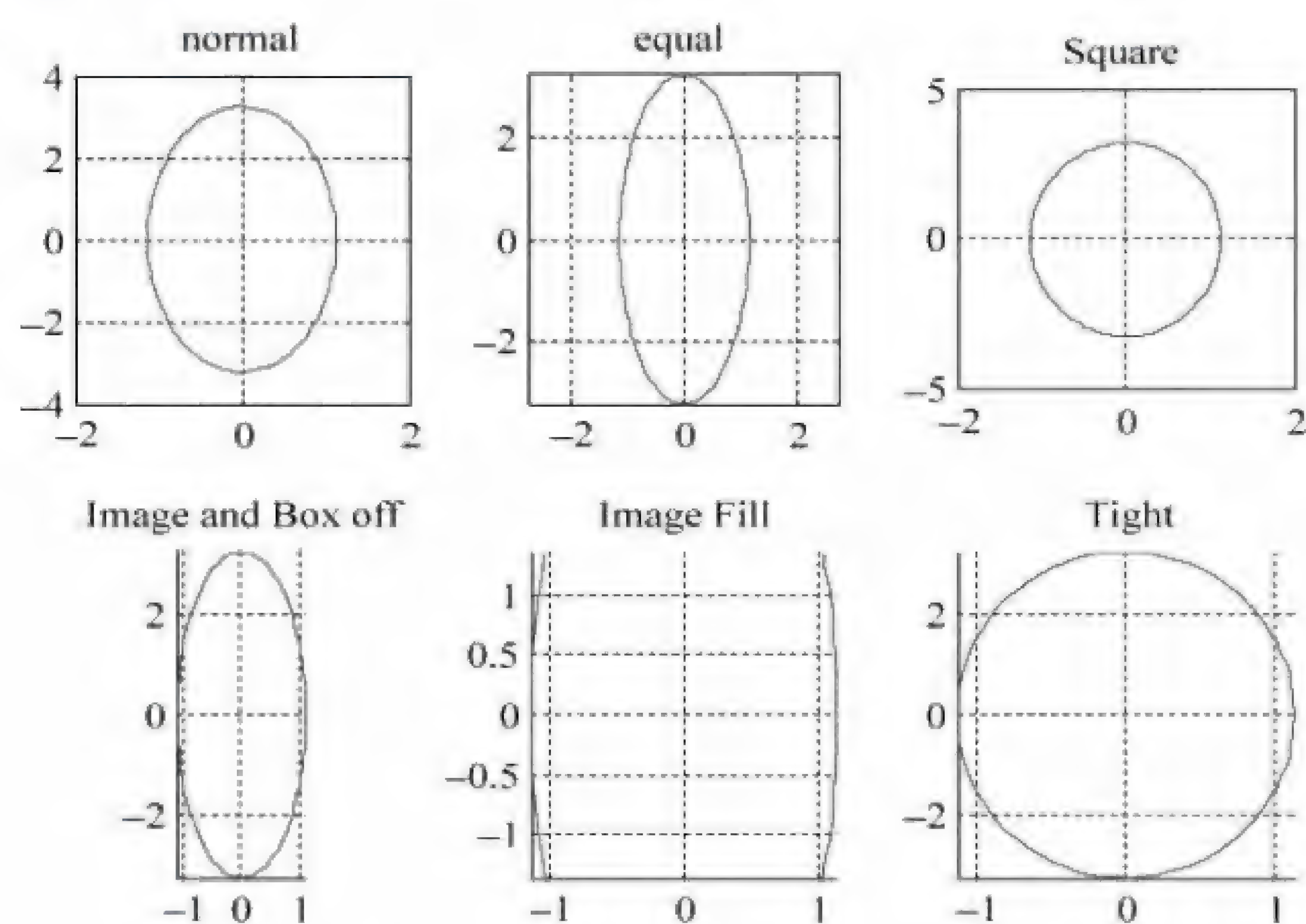


图 3-14　坐标轴变换对比图

【例 3-12】 将一个正弦函数的坐标轴由默认值修改为指定值。

解：依据题意编写如下代码：

```
clear all
clc
x = 0:0.03:3 * pi;
y = sin(x);
plot(x,y)
axis([0 3 * pi  - 2 2])
title('正弦波图形')
```

输出图形如图 3-15 所示。

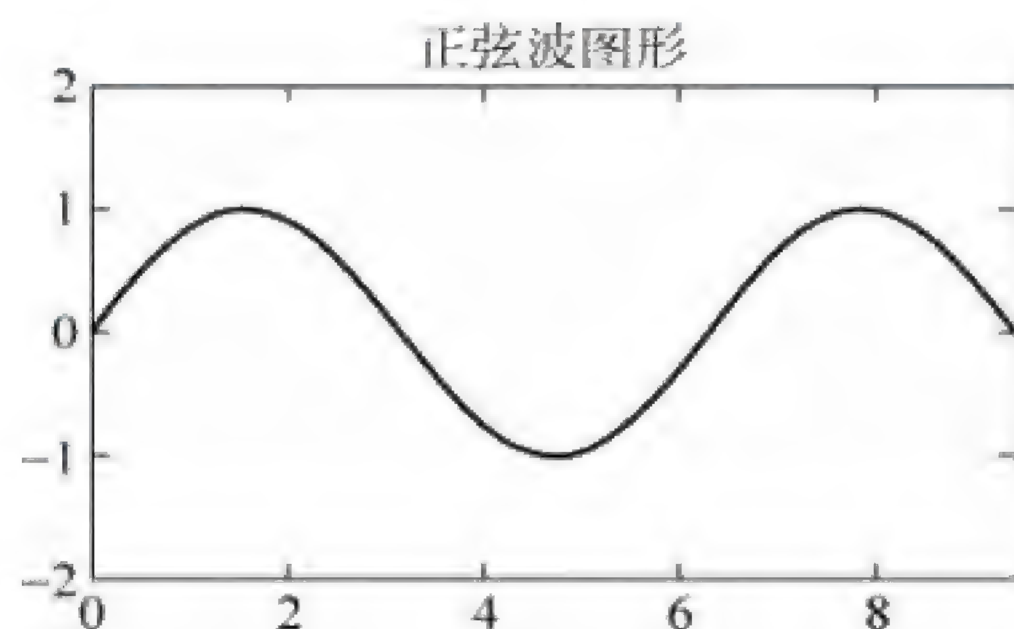


图 3-15 坐标轴调整示意图

2. 设置坐标框

使用 box 命令，可以开启或封闭二维图形的坐标框，其使用方法如下：

box：坐标形式在封闭和开启间切换。

box on：开启。

box off：封闭。

在实际使用过程中，系统默认为坐标框处于开启状态。

【例 3-13】 使用 box 命令，演示坐标框开启和封闭之间的区别。

解：依据题意编写如下代码：

```
clear all;
clc;
x = linspace( - 3 * pi,3 * pi);
y1 = sin(x);
y2 = cos(x);
figure
h = plot(x,y1,x,y2);
box on
```

输出有坐标框图形如图 3-16 所示。

在上面代码后面增加如下语句：

```
box off;
```

即可以看到如图 3-17 所示的无坐标框二维图。

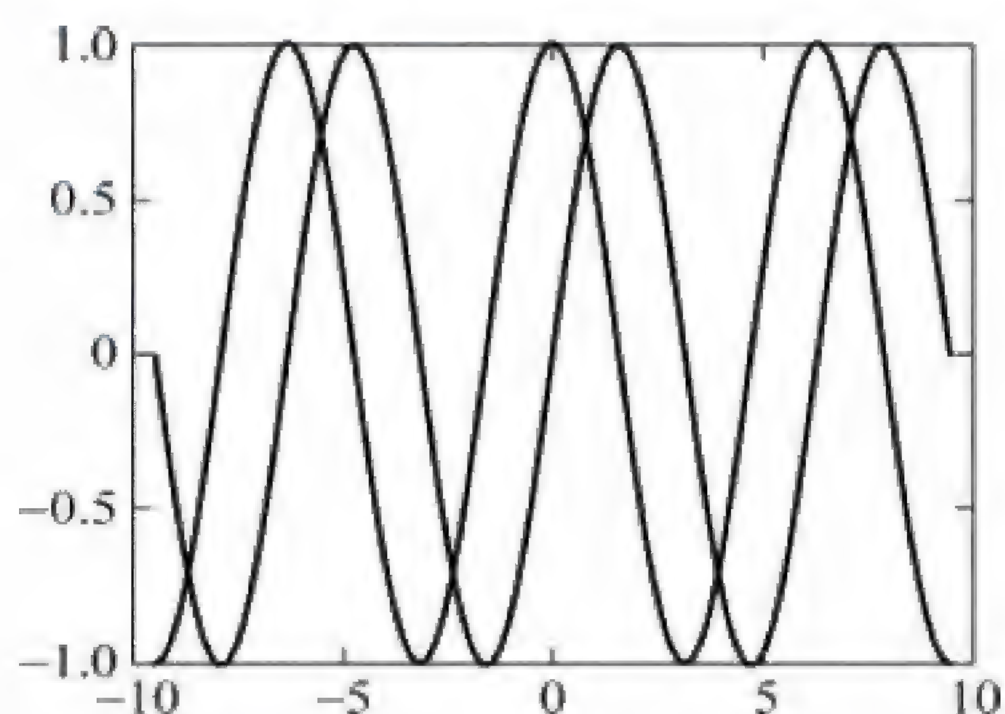


图 3-16　有坐标框的二维图

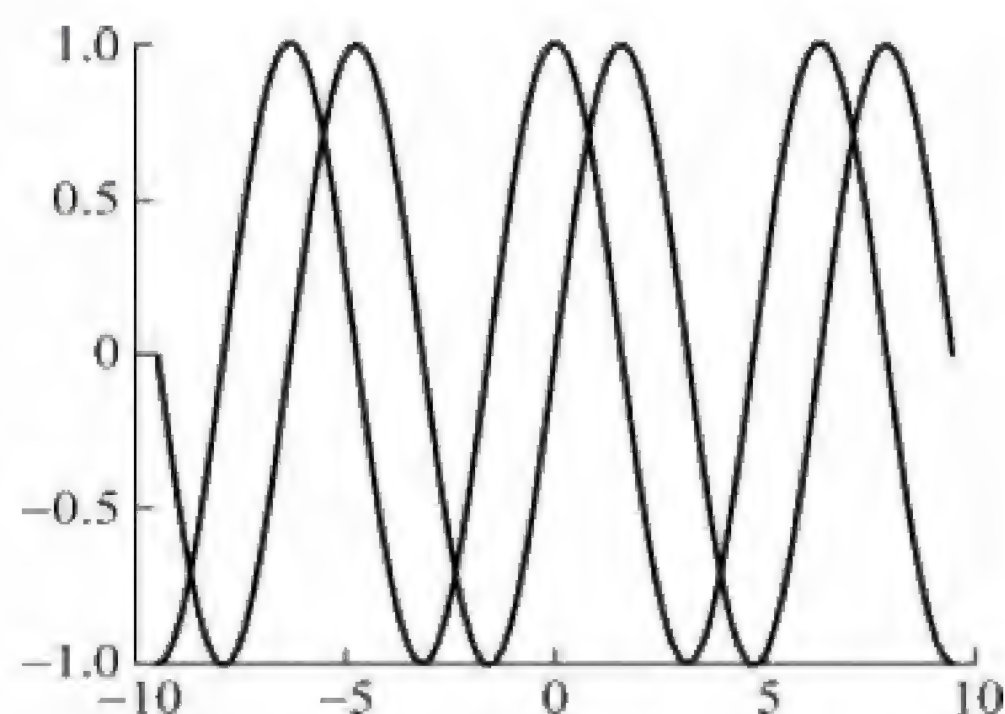


图 3-17　无坐标框的二维图

3. 图形标识

在 MATLAB 中增加标识可以使用 title 和 text 命令。其中，title 是将标识添加在固定位置，text 是将标识添加到用户指定位置。

使用 title('string')命令给绘制的图形加上固定位置的标题，xlabel ('string')命令和 ylabel ('string')命令分别给 x 轴和 y 轴加上标注。

例如，在 MATLAB 命令行窗口输入如下命令，可得到如图 3-18 所示的图形。

```
clear all
clc
x = 0:0.02:3 * pi;
y1 = 2 * sin(x);
y2 = cos(x);
plot(x,y1,x,y2, '--')
grid on;
xlabel ('弧度值')
ylabel ('函数值')
title('不同幅度的正弦与余弦曲线')
```

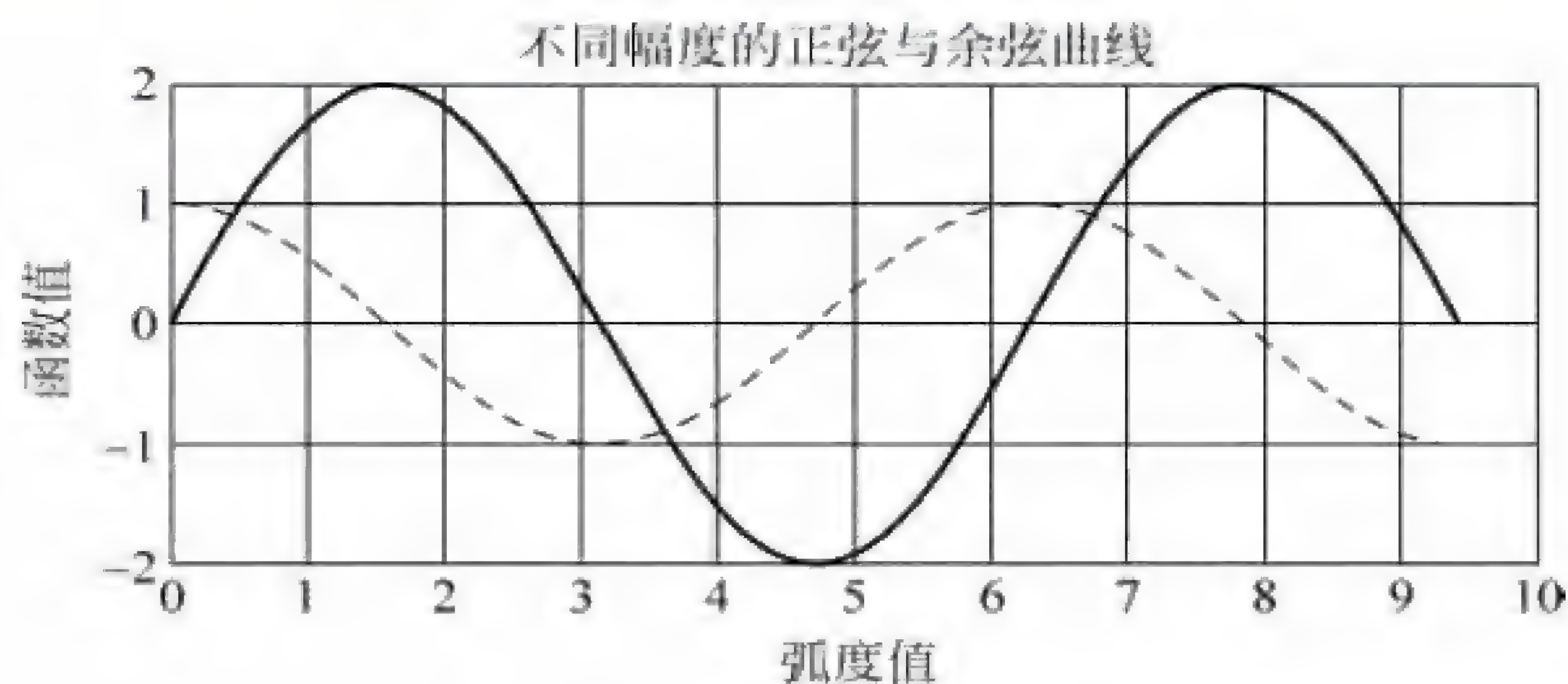


图 3-18　标识坐标轴名称

在 MATLAB 中，用户可以在图形的任意位置加注一串文本作为注释。在任意位置加注文本可以使用坐标轴确定文字位置的 text 命令，其使用格式如下：

```
text(x,y, 'string','option')
```

在图形的指定坐标位置(x,y)处，写出由 string 所给出的字符串。其中 x，y 坐标的单位是由后面的 option 选项决定的。如果不加选项，则 x，y 的坐标单位和图中一致；如果选项为'sc'，表示坐标单位是取左下角为(0,0)，右上角为(1,1)的相对坐标。

在画出图 3-18 所示图形后，继续输入如下命令：

```
text(0.4,0.8, '正弦曲线', 'sc')
text(0.7,0.8, '余弦曲线', 'sc')
```

得到如图 3-19 所示的图形。

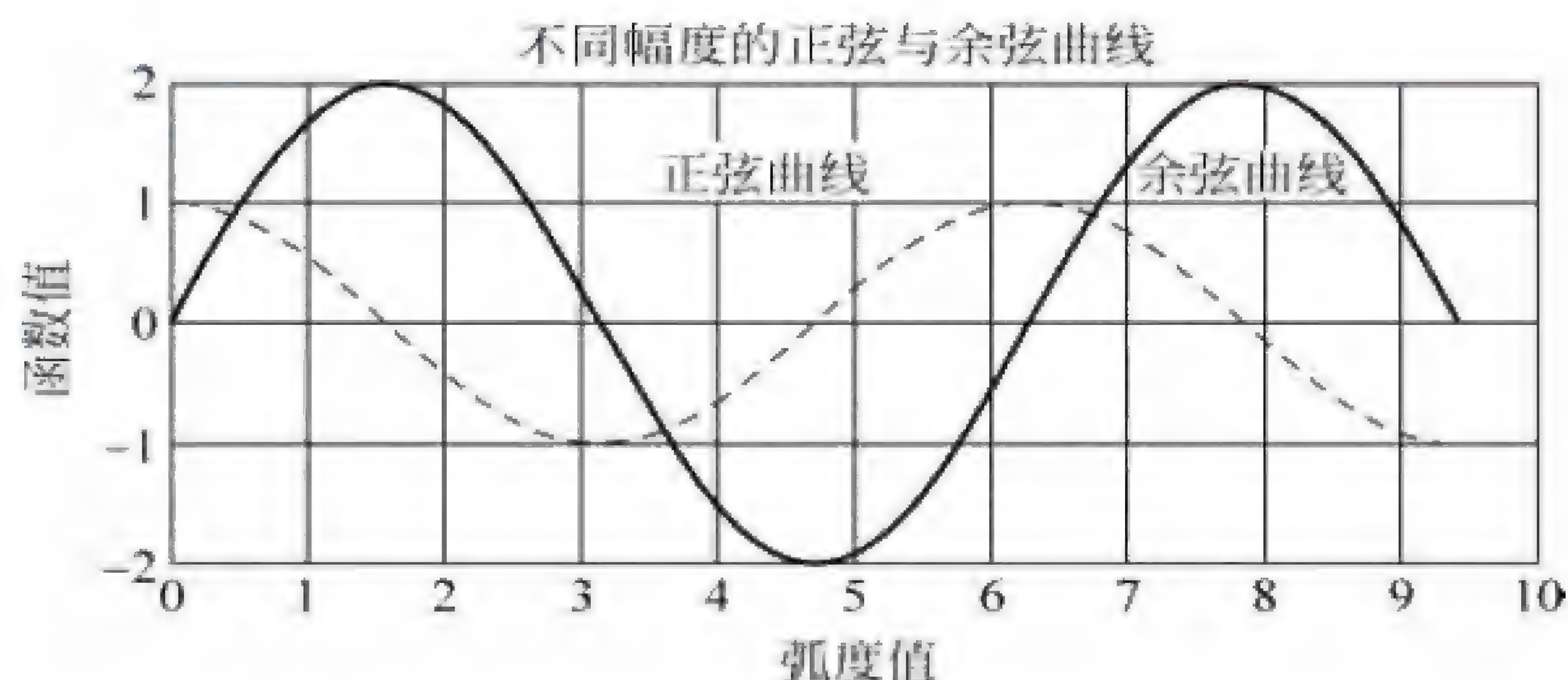


图 3-19 曲线加注名称

【例 3-14】 使用 text 命令，计算标注文字的位置。

解：依据题意编写如下 MATLAB 代码：

```
clear all
clc
t = 0 : 700;
hold on;
plot ( t, 0.35 * exp ( - 0.005 * t ) );
text ( 300,0.35 * exp ( - 0.005 * 300 ), '\bullet \leftarrow \fontname {times} 0.05 at t =
300' , 'FontSize' , 14 )
hold off ;
```

得到如图 3-20 所示的图形。

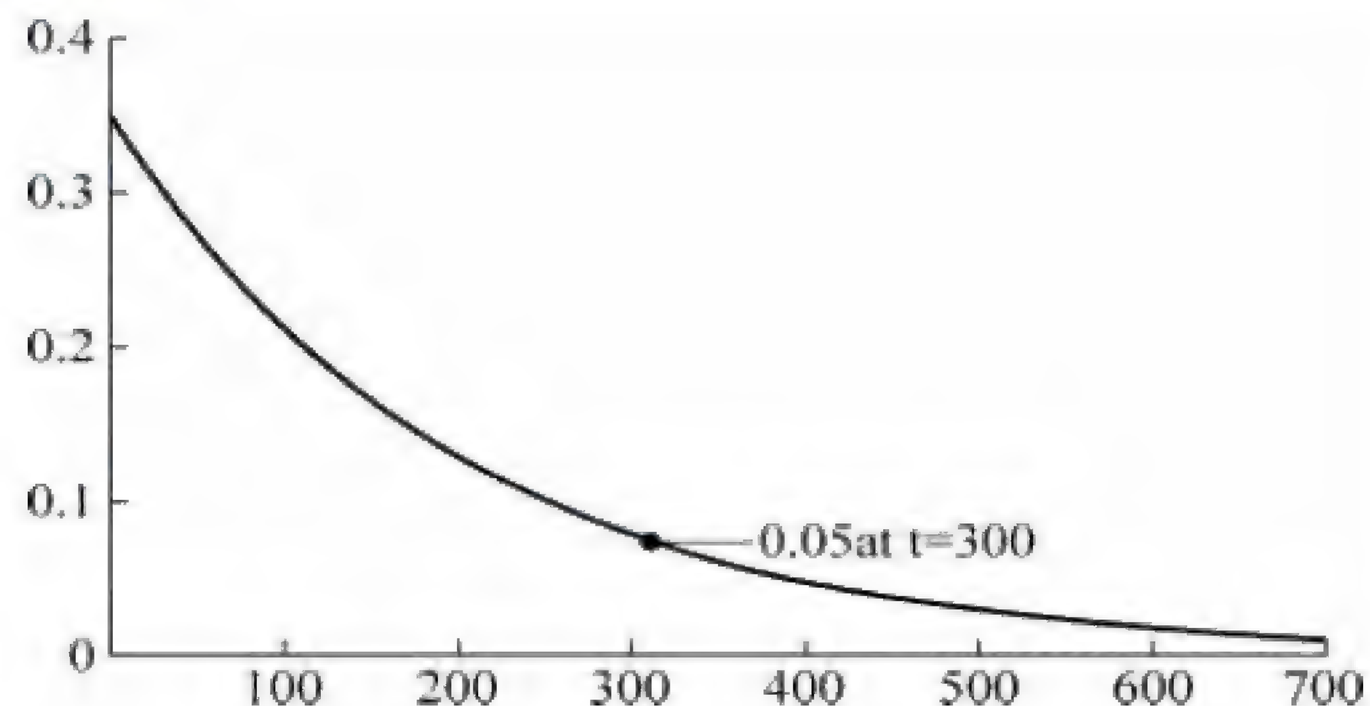


图 3-20 计算标注文字位置

【例 3-15】 使用 text 命令，绘制连续和离散数据图形，并对图形进行标识。

解：依据题意编写如下代码：

```
clear all
clc
x = linspace ( 0, 2 * pi, 60 ) ;
a = sin ( x ) ;
b = cos ( x );
hold on
stem_handles = stem ( x, a + b );
plot_handles = plot ( x, a, '-r', x, b, '-g');
xlabel ('Time in \ musecs' )
ylabel ('Magnitude' )
title ('Linear Combination of Two Functions ')
legend_handles = [ stem_handles ; plot_handles ];
legend ( legend_handles, 'a + b', 'a = sin ( x )', 'b = cos ( x )')
```

得到如图 3-21 所示的详细文字标识图形。

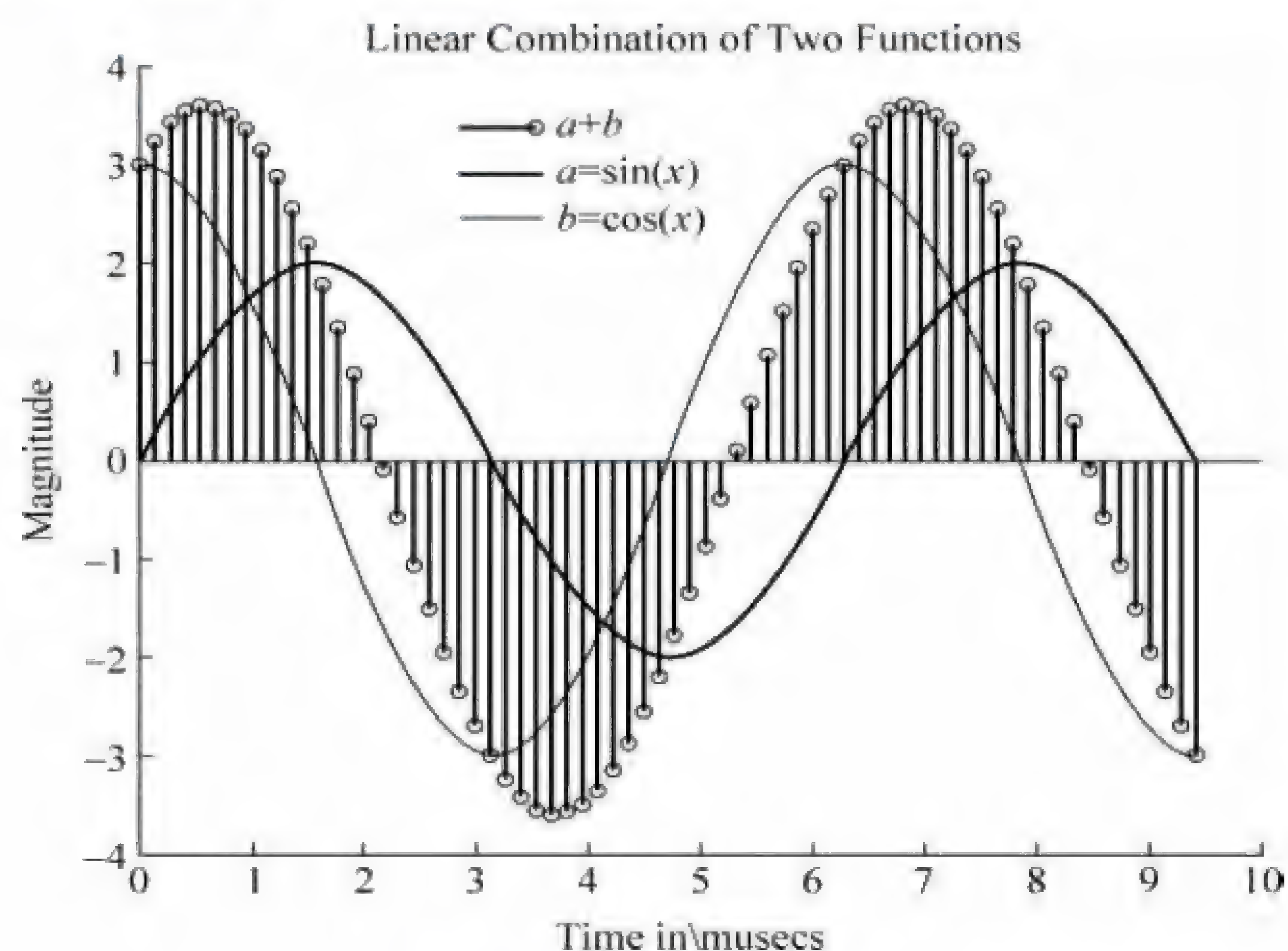


图 3-21 详细文字标识

【例 3-16】 使用 text 命令，绘制包括不同统计量的标注说明。

解：依据题意编写如下代码：

```
clear all
clc
x = 0:0.3:15;
b = bar(rand(10,5),'stacked'); colormap(summer); hold on
x = plot(1:10,5 * rand(10,1),'marker','square','markersize',12, 'markeredgecolor','y','
markerfacecolor',[.6 0 .6],'linestyle','-','color','r','linewidth',2);
hold off
```

```
legend([b,x],'Carrots','Peas','Peppers','Green Beans', 'Cucumbers','Eggplant')

b = bar(rand(10,5),'stacked');
colormap(summer);
hold on
x = plot(1:10,5 * rand(10,1),'marker','square','markersize',12, 'markeredgecolor','y',
'markerfacecolor',[.6 0 .6],'linestyle','-','color','r','linewidth',2);
hold off
legend([b,x],'Carrots','Peas','Peppers','Green Beans','Cucumbers','Eggplant')
```

得到如图 3-22 所示包括不同统计量的标注说明图形。

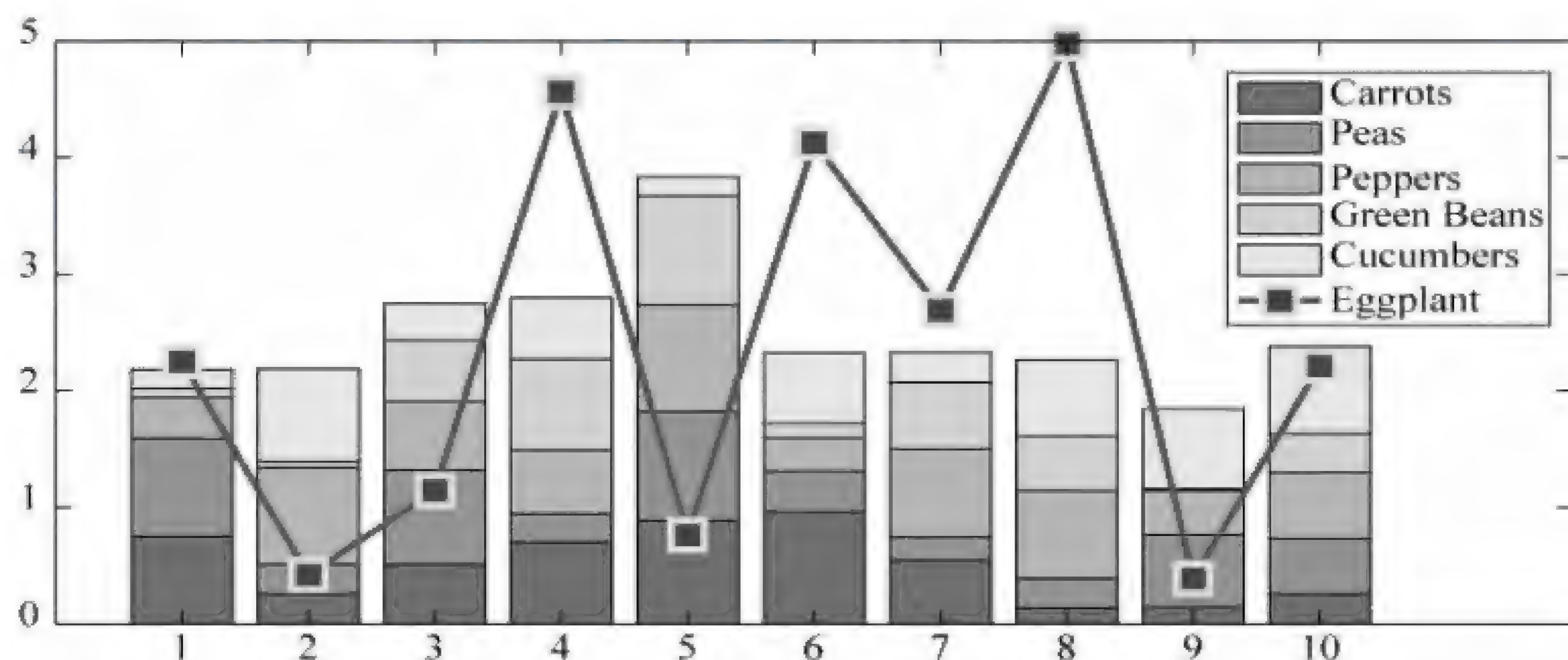


图 3-22　包括不同统计量的标注说明图形

4．图案填充

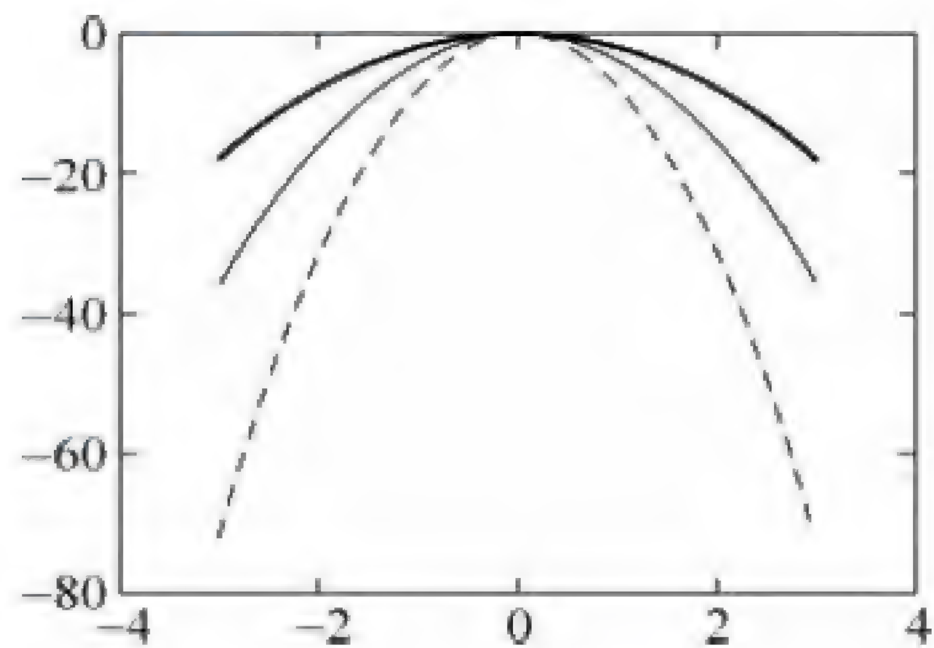


图 3-23　原始图形

MATLAB 除了可以直接画出单色二维图之外，还可以使用 patch 函数在指定的两条曲线和水平轴所包围的区域填充指定的颜色，其使用格式如下：

patch(x, y, [r g b])：[r g b]中的 r 表示红色，g 表示绿色，b 表示蓝色。

【例 3-17】 使用函数在图 3-23 中的两条实线之间填充红色，并在两条虚线之间填充黑色。

解：依据题意编写如下代码：

```
clear all
clc
x = -1:0.01:1;
y = -1. * x. * x;
plot(x,y,'-','LineWidth',1)
XX = x;
YY = y;
hold on
```

```
y = - 2. * x. * x;
plot(x,y,'r-','LineWidth',1)
hold on
XX = [XX x(end: - 1:1)];
YY = [YY y(end: - 1:1)];
patch(XX,YY,'r')

y = - 4. * x. * x;
plot(x,y,'g--','LineWidth',1)
XX = x;
YY = y;
hold on
y = - 8. * x. * x;
plot(x,y,'k--','LineWidth',1)
XX = [XX x(end: - 1:1)];
YY = [YY y(end: - 1:1)];
patch(XX,YY,'b')
```

得到图形如图 3-24 所示。

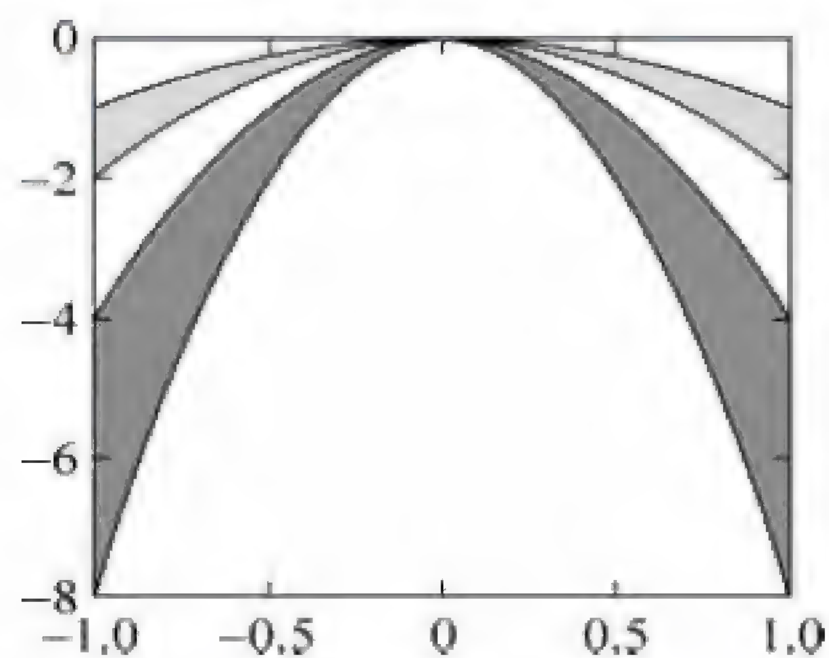


图 3-24　颜色填充后图形

3.2.3　子图绘制法

在一个图形窗口用函数 subplot 可以同时画出多个子图形，其调用格式主要有以下几种：

1. subplot(m,n,p)

将当前图形窗口分成 m×n 个子窗口，并在第 x 个子窗口建立当前坐标平面。子窗口按从左到右，从上到下的顺序编号，如图 3-25 所示。如果 p 为向量，则以向量表示的位置建立当前子窗口的坐标平面。

2. subplot(m,n,p,'replace')

按图 3-25 所示建立当前子窗口的坐标平面时，若指定位置已经建立了坐标平面，则以新建的坐标平面代替原本的坐标平面。

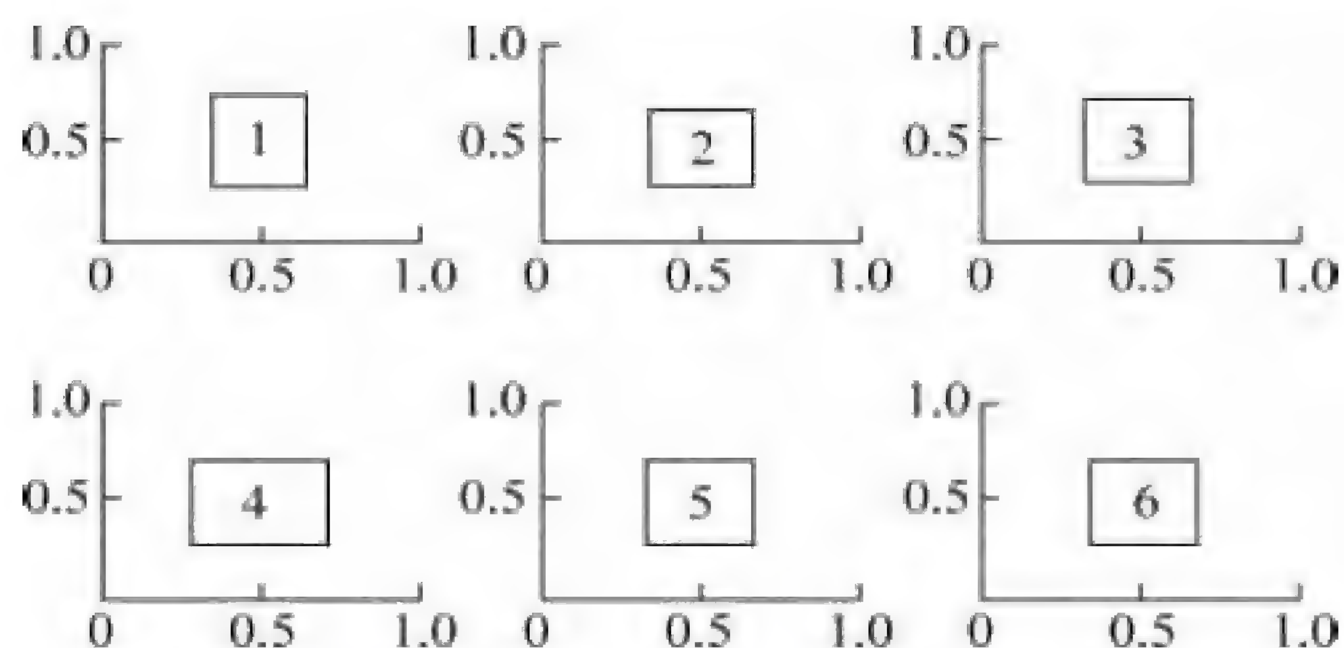


图 3-25 子图位置示意图

3. subplot(h)

指定当前子图坐标平面的句柄 h，h 为按 mnp 排列的整数，如在图 3-25 所示的子图中 h=232，表示第 2 个子图坐标平面的句柄。

4. subplot('Position',[left bottom width height])

在指定的位置建立当前子图坐标平面，它把当前图形窗口看成是 1.0×1.0 的平面，所以 left、bottom、width、height 分别在(0,1)的范围内取值，表示所创建的当前子图坐标平面距离图形窗口左边、底边的长度以及所建子图坐标平面的宽度和高度。

5. h = subplot(…)

创建当前子图坐标平面时，同时返回其句柄。值得注意的是函数 subplot 只是创建子图坐标平面，当在该坐标平面内绘制子图时，仍然需要使用 plot 函数或其他绘图函数。

【例 3-18】 用 subplot 函数画一个子图，要求两行两列共 4 个子窗口，且分别画出正弦、余弦、正切、余切函数曲线。

解：依据题意编写如下代码：

```
clear all
clc
x = -4:0.01:4;
subplot(2,2,1);
plot(x,sin(x));                    % 画 sin(x)
xlabel('x');
ylabel('y');
title('sin(x)')
subplot(2,2,2);
plot(x,cos(x));                    % 画 cos(x)
xlabel('x');
ylabel('y');
title('cos(x)');
subplot(2,2,3);
x = (-pi/2)+0.01:0.01:(pi/2)-0.01;
plot(x,tan(x));                    % 画 tan(x)
```

```
xlabel('x');
ylabel('y');
title('tan x');
subplot(2,2,4);
x = 0.01:0.01:pi - 0.01;
plot(x,cot(x));
xlabel('x');
ylabel('y');
title('cot x');          % 画 cot(x)
```

输出图形如图 3-26 所示。

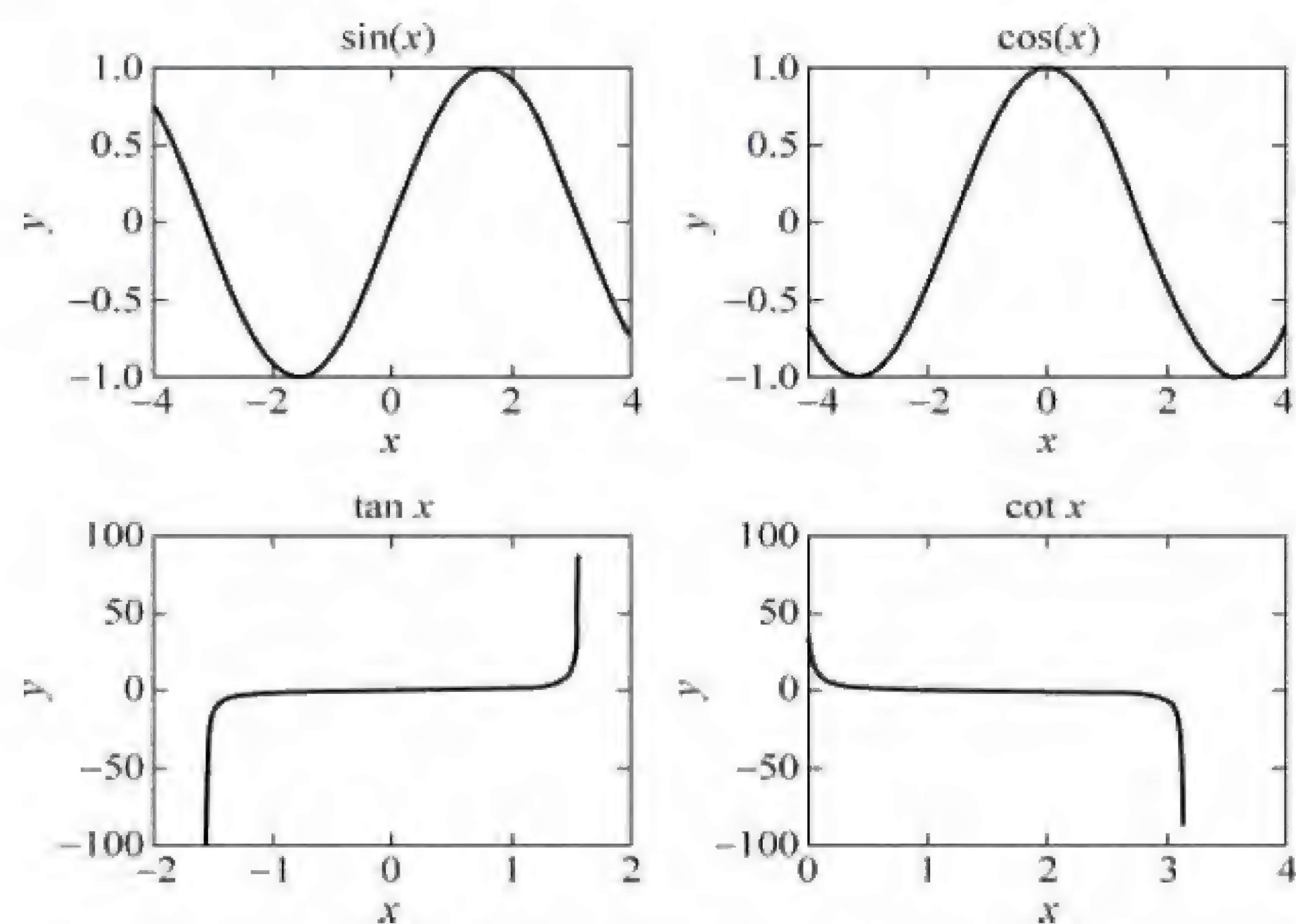


图 3-26　子图

【例 3-19】 用 subplot 函数画一个子图，要求两行两列共 4 个子窗口，且分别显示四种不同的曲线图像。

解：依据题意编写如下代码：

```
clear all
clc
t = 0:pi/10:3 * pi;
[ x, y ] = meshgrid ( t );

subplot ( 2, 2, 1 )
plot ( sin ( t ), cos ( t ) )
axis equal

subplot ( 2, 2, 2 )
z = sin ( x ) + 2 * cos ( y );
plot ( t, z )
axis ( [ 0 2 * pi  - 2 2 ] )
```

```
subplot ( 2, 2, 3 )
z = 2 * sin ( x ). * cos ( y );
plot ( t, z )
axis ( [ 0 2 * pi - 1 1 ] )

subplot ( 2, 2, 4 )
z = ( sin ( x ).^2 ) - ( cos ( y ).^2);
plot ( t, z )
axis ( [ 0 2 * pi - 1 1 ] )
```

输出图形如图 3-27 所示。

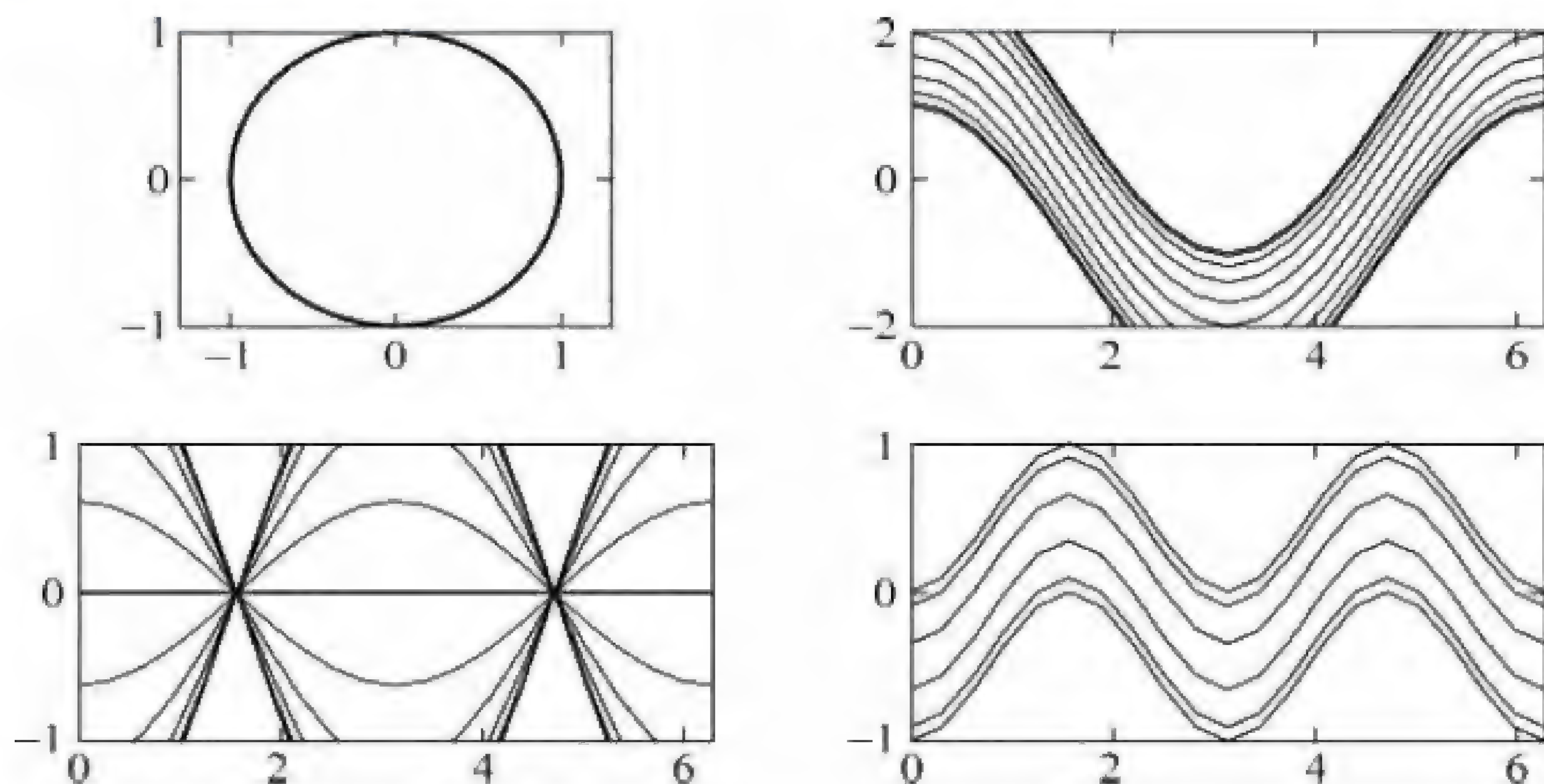


图 3-27　子图图形

3.2.4　二维绘图的经典应用

【例 3-20】　利用 MATLAB 绘图函数，绘制模拟电路演示过程，要求电路中有蓄电池、开关和灯，开关默认处于不闭合状态。当开关闭合后，灯变亮。

解：在 MATLAB 命令行窗口中输入：

```
clear all
clc
figure('name','模拟电路图');
axis([ - 4,14,0,10]);                        % 建立坐标系
hold on                                      % 保持当前图形的所有特性
axis('off');                                 % 关闭所有轴标注和控制
 % 绘制蓄电池的过程
fill([ - 1.5, - 1.5,1.5,1.5],[1,5,5,1],[0.5,1,1]);
fill([ - 0.5, - 0.5,0.5,0.5],[5,5.5,5.5,5],[0,0,0]);
text( - 0.5,1.5,'—');
text( - 0.5,3,'电池');
text( - 0.5,4.5,' + ');                      % 绘制导电线路的过程
plot([0;0],[5.5;6.7],'color','r','linestyle',' - ','linewidth',4);
                                             % 绘制二维图形线竖实心红色
```

```
plot([0;4],[6.7;6.7],'color','r','linestyle','-','linewidth',4); %绘制二维图形线,实心
                                                                  红色为导线
a=line([4;5],[6.7;7.7],'color','b','linestyle','-','linewidth',4,'erasemode','xor');
%画开关蓝色
plot([5.2;9.2],[6.7;6.7],'color','r','linestyle','-','linewidth',4);%绘制图导线为红色
plot([9.2;9.2],[6.7;3.7],'color','r','linestyle','-','linewidth',4);%绘制图导线竖线为
                                                                    红线
plot([9.2;9.7],[3.7;3.7],'color','r','linestyle','-','linewidth',4); %绘制图导线横线为
                                                                    红色
plot([0;0],[1;0],'color','r','linestyle','-','linewidth',4);  %如上画红色竖线
plot([0;10],[0;0],'color','r','linestyle','-','linewidth',4); %如上画横线
plot([10;10],[0;3],'color','r','linestyle','-','linewidth',4);%画竖线
%绘制灯泡的过程
fill([9.8,10.2,9.7,10.3],[3,3,3.3,3.3],[0 0 0]);               %确定填充范围
plot([9.7,9.7],[3.3,4.3],'color','b','linestyle','-','linewidth',0.5);  %绘制灯泡外形
                                                                       线为蓝色
plot([10.3,10.3],[3.3,4.45],'color','b','linestyle','-','linewidth',0.5);
%绘制圆
x=9.7:pi/50:10.3;
plot(x,4.3+0.1*sin(40*pi*(x-9.7)),'color','b','linestyle','-','linewidth',0.5);
t=0:pi/60:2*pi;
plot(10+0.7*cos(t),4.3+0.6*sin(t),'color','b');
%下面是箭头及注释的显示
text(4.5,10,'电流方向');
line([4.5;6.6],[9.4;9.4],'color','r','linestyle','-','linewidth',4,'erasemode','xor');
  %绘制箭头横线
line(6.7,9.4,'color','b','linestyle','-','erasemode','xor','markersize',10);%绘制箭头三
                                                                           角形
pause(1);
%绘制开关闭合的过程
t=0;
y=7.6;
while y>6.6                                          %电路总循环控制开关动作条件
x=4+sqrt(2)*cos(pi/4*(1-t));
y=6.7+sqrt(2)*sin(pi/4*(1-t));
set(a,'xdata',[4;x],'ydata',[6.7;y]);
drawnow;
t=t+0.1;
end
%绘制开关闭合后模拟大致电流流向的过程
pause(1);
light=line(10,4.3,'color','y','marker','.','markersize',40,'erasemode','xor');  %画灯丝
发出的光:黄色
%画电流的各部分
h=line([1;1],[5.2;5.6],'color','r','linestyle','-','linewidth',4,'erasemode','xor');
g=line(1,5.7,'color','b','linestyle','-','erasemode','xor','markersize',10);
%给循环初值
t=0;
m2=5.6;
```

```
n = 5.6;
while n < 6.5;              % 确定电流竖向循环范围
m = 1;
n = 0.05 * t + 5.6;
set(h,'xdata',[m;m],'ydata',[n - 0.5;n - 0.1]);
set(g,'xdata',m,'ydata',n);
t = t + 0.01;
drawnow;
end
t = 0;
while t < 1;                % 在转角处的停顿时间
m = 1.2 - 0.2 * cos((pi/4) * t);
n = 6.3 + 0.2 * sin((pi/4) * t);
set(h,'xdata',[m - 0.5;m - 0.1],'ydata',[n;n]);
set(g,'xdata',m,'ydata',n);
t = t + 0.05;
drawnow;
end
t = 0;
while t < 0.4               % 在转角后的停顿时间
t = t + 0.5;
g = line(1.2,6.5,'color','b','linestyle','^','markersize',10,'erasemode','xor');
g = line(1.2,6.5,'color','b','linestyle','>','markersize',10,'erasemode','xor');
set(g,'xdata',1.2,'ydata',6.5);
drawnow;
end
pause(0.5);
t = 0;
while m < 7                 % 确定第二个箭头的循环范围
m = 1.1 + 0.05 * t;
n = 6.5;
set(g,'xdata',m + 0.1,'ydata',6.5);
set(h,'xdata',[m - 0.4;m],'ydata',[6.5;6.5]);
t = t + 0.05;
drawnow;
end
t = 0;
while t < 1                 % 在转角后的停顿时间
m = 8.1 + 0.2 * cos(pi/2 - pi/4 * t);
n = 6.3 + 0.2 * sin(pi/2 - pi/4 * t);
set(g,'xdata',m,'ydata',n);
set(h,'xdata',[m;m],'ydata',[n + 0.1;n + 0.5]);
t = t + 0.05;
drawnow;
end
t = 0;
while t < 0.4               % 在转角后的停顿时间
t = t + 0.5;
 % 绘制第三个箭头
```

```
g=line(8.3,6.3,'color','b','linestyle','>','markersize',10,'erasemode','xor');
g=line(8.3,6.3,'color','b','linestyle','v','markersize',10,'erasemode','xor');
set(g,'xdata',8.3,'ydata',6.3);
drawnow;
end

pause(0.5);
t=0;
while n>1                    %确定箭头的运动范围
m=8.3;
n=6.3-0.05*t;
set(g,'xdata',m,'ydata',n);
set(h,'xdata',[m;m],'ydata',[n+0.1;n+0.5]);
t=t+0.04;
drawnow;
end
t=0;
while t<1                    %箭头的起始时间
m=8.1+0.2*cos(pi/4*t);
n=1-0.2*sin(pi/4*t);
set(g,'xdata',m,'ydata',n);
set(h,'xdata',[m+0.1;m+0.5],'ydata',[n;n]);
t=t+0.05;
drawnow;
end
t=0;
while t<0.5
t=t+0.5;
%绘制第四个箭头
g=line(8.1,0.8,'color','b','linestyle','v','markersize',10,'erasemode','xor');
g=line(8.1,0.8,'color','b','linestyle','<','markersize',10,'erasemode','xor');
set(g,'xdata',8.1,'ydata',0.8);
drawnow;
end
pause(0.5);
t=0;
while m>1.1                  %箭头的运动范围
m=8.1-0.05*t;
n=0.8;
set(g,'xdata',m,'ydata',n);
set(h,'xdata',[m+0.1;m+0.5],'ydata',[n;n]);
t=t+0.04;
drawnow;
end
t=0;
while t<1                    %停顿时间
m=1.2-0.2*sin(pi/4*t);
n=1+0.2*cos(pi/4*t);
set(g,'xdata',m,'ydata',n);
set(h,'xdata',[m;m+0.5],'ydata',[n-0.1;n-0.5]);
```

```
t = t + 0.05;
drawnow;
end
t = 0;
while t < 0.5                    % 画第五个箭头
t = t + 0.5;
g = line(1,1,'color','b','linestyle','<','markersize',10,'erasemode','xor');
g = line(1,1,'color','b','linestyle','^','markersize',10,'erasemode','xor');
set(g,'xdata',1,'ydata',1);
drawnow;
end
t = 0;
while n < 6.2
m = 1;
n = 1 + 0.05 * t;
set(g,'xdata',m,'ydata',n);
set(h,'xdata',[m;m],'ydata',[n - 0.5;n - 0.1]);
t = t + 0.04;
drawnow;
end
%绘制开关断开后的情况
t = 0;
y = 6.6;
while y < 7.6                    % 开关的断开
x = 4 + sqrt(2) * cos(pi/4 * t);
y = 6.7 + sqrt(2) * sin(pi/4 * t);
set(a,'xdata',[4;x],'ydata',[6.7;y]);
drawnow;
t = t + 0.1;
end
pause(0.2);                      % 开关延时作用
nolight = line(10,4.3,'color','y','marker','.','markersize',40,'erasemode','xor');
```

代码运行后，得到模拟电路图形如图 3-28 所示。

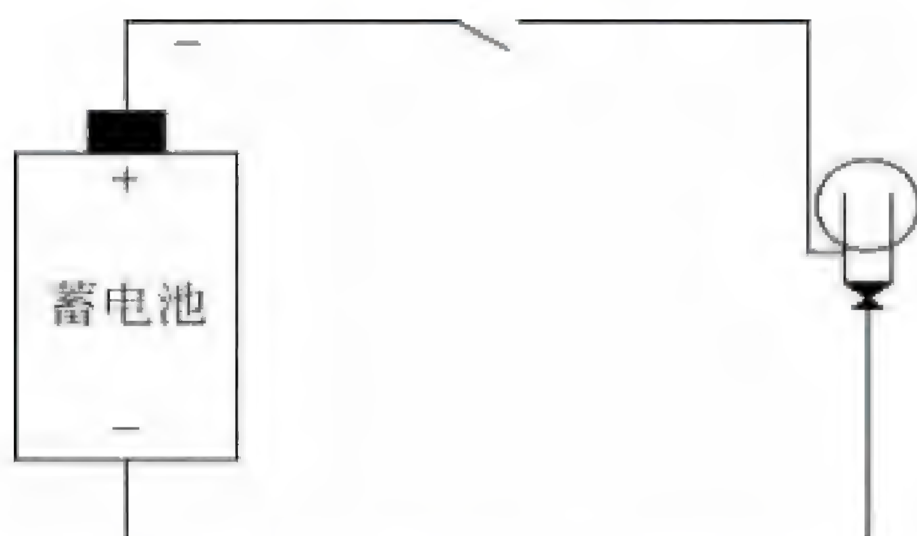


图 3-28　模拟电路演示图

3.3　三维绘制

MATLAB 中的三维图形包括三维折线、曲线图及三维曲面图等。创建三维图形和创建二维图形的过程类似，都包括数据准备、绘图区选择、绘图、设置、标注以及图形的打印或输出。不过，三维图形能够设置和标注更多的元素，如颜色过渡、光照和视角等。

3.3.1 三维绘图基本命令

绘制二维折线或曲线时可以使用 plot 命令。与这条命令类似，MATLAB 也提供了一个绘制三维折线或曲线的基本命令 plot3，其格式如下：

```
plot3(x1,y1,z1,option1,x2,y2,z2,option2, ….)
```

plot3 命令以 x1，y1，z1 所给出的数据分别为 x，y，z 坐标值，option1 为选项参数，以逐点连折线的方式绘制 1 个三维折线图形；同时，以 x2，y2，z2 所给出的数据分别为 x，y，z 坐标值，option2 为选项参数，以逐点折线的方式绘制另一个三维折线图形。

plot3 命令的功能及使用方法与 plot 命令的功能及使用方法相类似，它们的区别在于前者绘制出的是三维图形。

plot3 命令参数的含义与 plot 命令的参数含义相类似，它们的区别在于前者多了一个 z 方向上的参数。同样，各个参数的取值情况及其操作效果也与 plot 命令相同。

plot3 命令使用的是以逐点连线的方法来绘制三维折线的，当各个数据点的间距较小时，也可利用它来绘制三维曲线。

【例 3-21】 绘制三维曲线示例。

解：依据题意编写如下代码：

```
clear all
clc
t=0:0.4:40;
figure(1)
subplot(2,2,1);
plot3(sin(t),cos(t),t);                    %画三维曲线
grid,
text(0,0,0,'0');                           %在 x=0,y=0,z=0 处标记"0"
title('Three Dimension');
xlabel('sin(t)'),
ylabel('cos(t)'),
zlabel('t');
subplot(2,2,2);plot(sin(t),t);
grid
title('x-z plane');                        %三维曲线在 x-z 平面的投影
xlabel('sin(t)'),
ylabel('t');
subplot(2,2,3);
plot(cos(t),t);
grid
title('y-z plane');                        %三维曲线在 y-z 平面的投影
xlabel('cos(t)'),
ylabel('t');
subplot(2,2,4);
plot(sin(t),cos(t));
```

```
title('x-y plane');           % 三维曲线在 x-y 平面的投影
xlabel('sin(t)'),
ylabel('cos(t)');
grid
```

输出图形如图 3-29 所示。

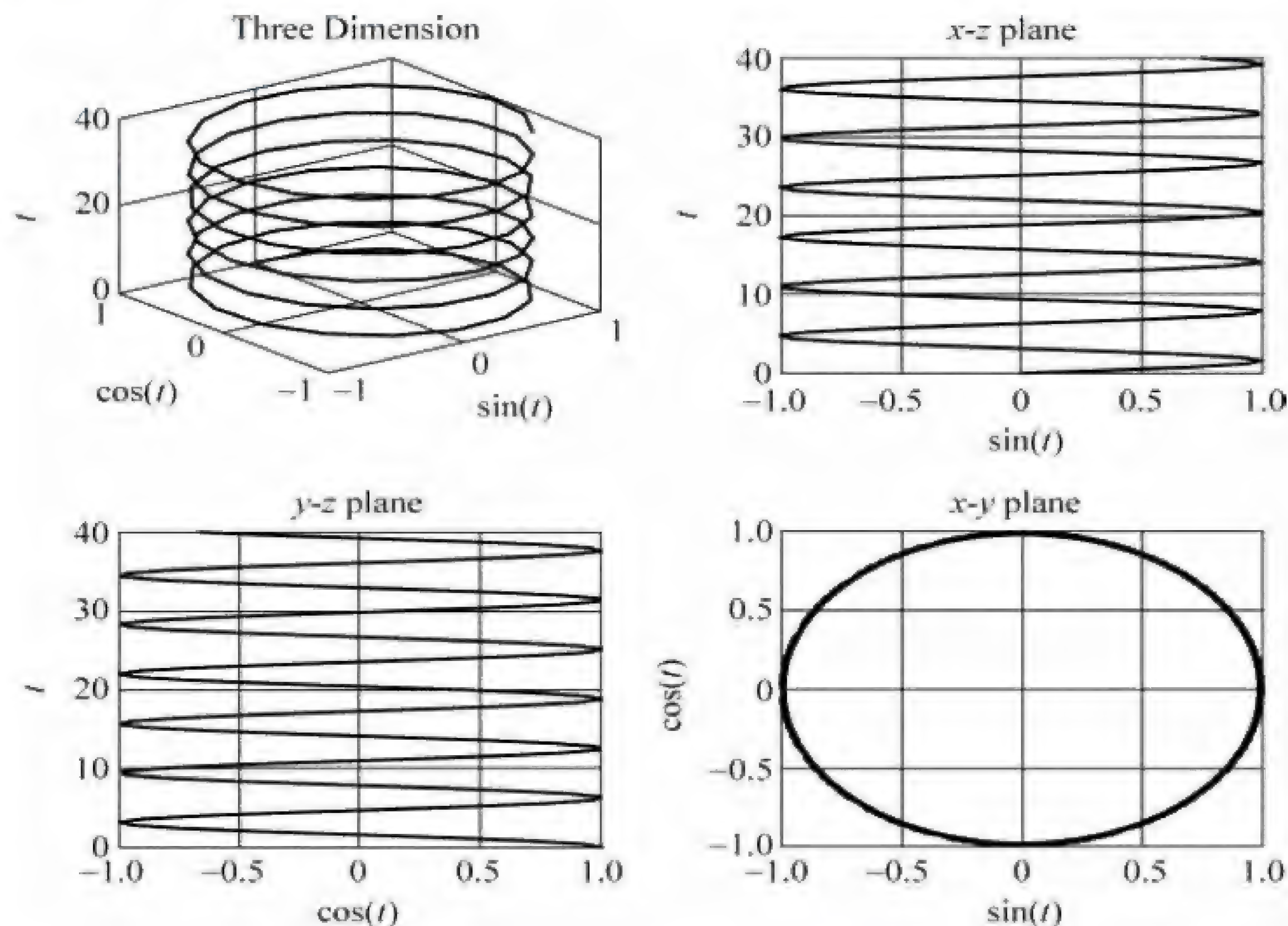


图 3-29　三维曲线及三个平面上的投影

【例 3-22】　绘制函数 $z=\sqrt{x^2+2y^2}$ 的图形，其中 $(x,y)\in[-7,7]$。

解：使用 MATLAB 作图的程序代码为

```
clear all
clc
x = -7:0.1:7;
y = -7:0.1:7;
[X,Y] = meshgrid(x,y);              % 将向量 x,y 指定的区域转化为矩阵 X,Y
Z = sqrt(X.^2 + 2 * Y.^2);          % 产生函数值 Z
mesh(X,Y,Z)
```

输出图形如图 3-30 所示。

【例 3-23】　利用 plot3 绘制 $x=2\sin(t)$、$y=3\cos(t)$ 三维螺旋线。

解：在命令行窗口编写 MATLAB 代码：

```
clear all
clc
t = 0:pi/100:7 * pi;
x = 2 * sin(t);
```

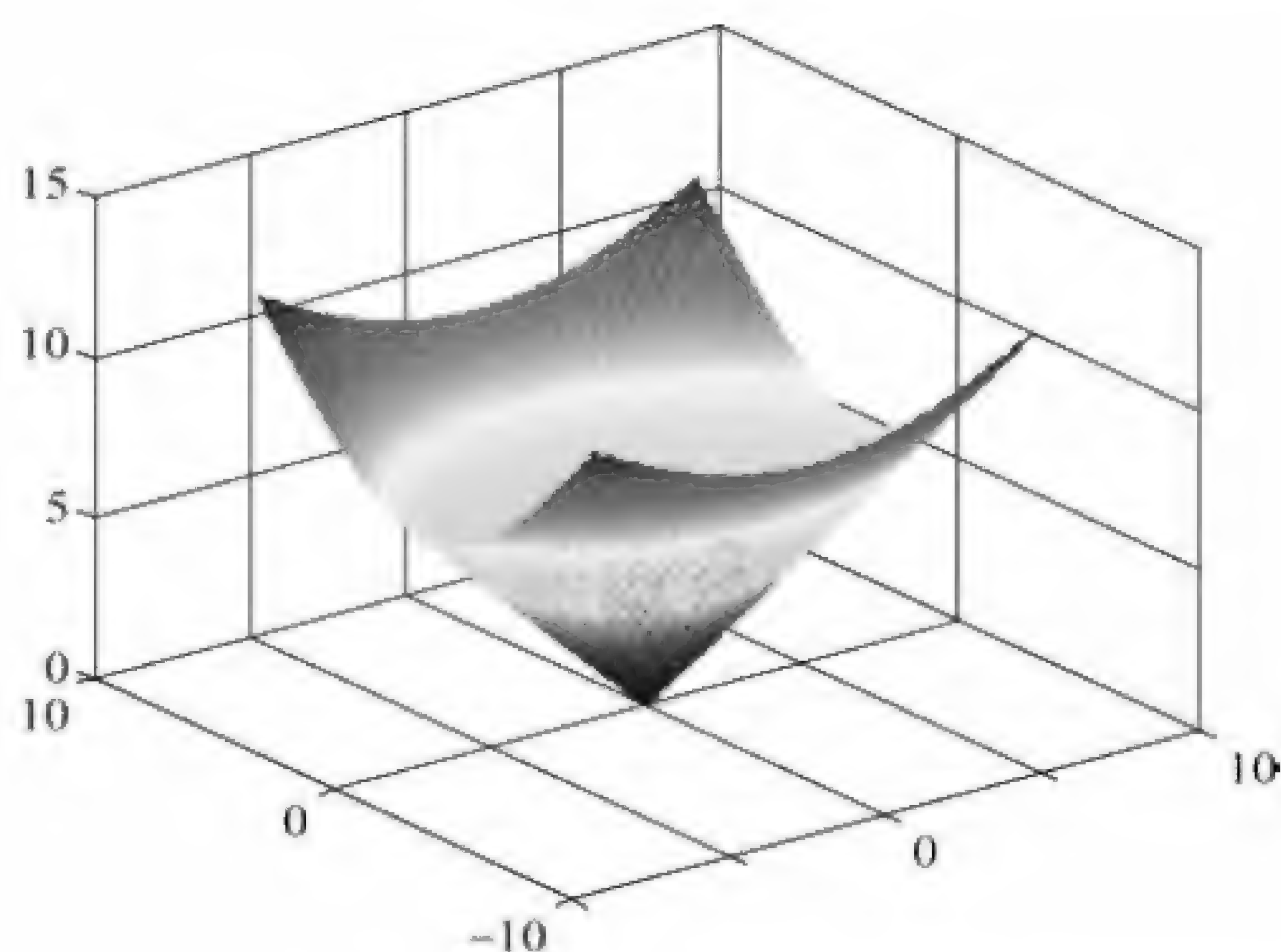


图 3-30　函数 $z=\sqrt{x^2+2y^2}$ 图形

```
y = 3 * cos(t);
z = t;
plot3(x,y,z)
```

执行程序后，得到如图 3-31 所示图形。

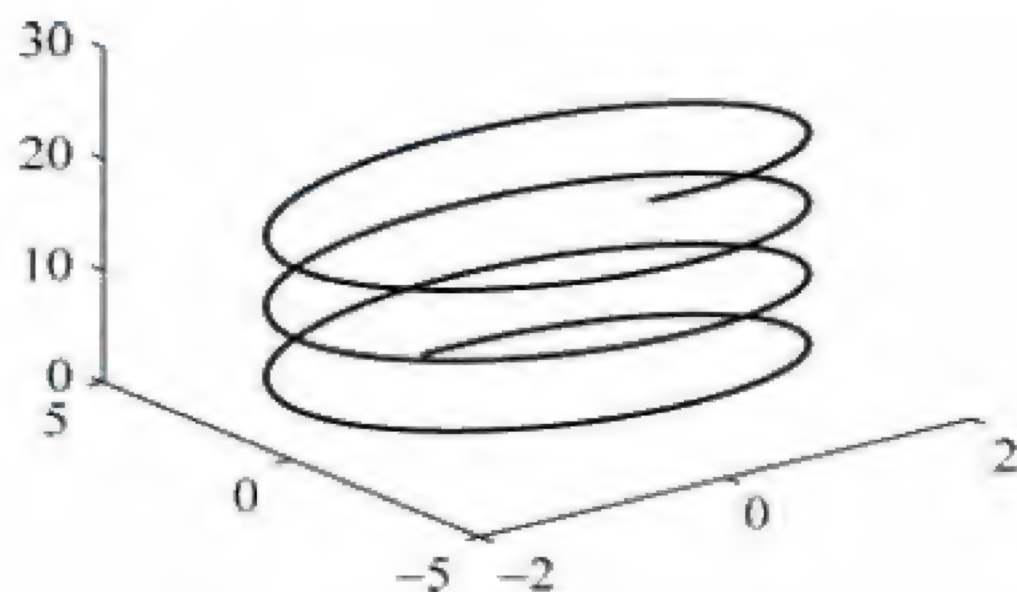


图 3-31　三维螺旋线图形

【例 3-24】 利用 plot3 绘制 $z=3x(-x^3-2y^2)$ 三维线条图形。

解：编写 MATLAB 代码如下：

```
clear all
clc
[X,Y] = meshgrid([ - 4:0.1:4]);
Z = 3 * X. * ( - X.^3 - 2 * Y.^2);
plot3(X,Y,Z,'b')
```

执行程序后，显示结果如图 3-32 所示。

在 MATLAB 中，可用函数 surf、surfc 来绘制三维曲面图。其调用格式如下：

surf(Z)：以矩阵 Z 指定的参数创建一渐变的三维曲面，坐标 x=1∶n，y=1∶m，其中[m，n]=size(Z)。

surf(X，Y，Z)：以 Z 确定的曲面高度和颜色，按照 X、Y 形成的格点矩阵，创建一渐变的三维曲面。X、Y 可以为向量或矩阵，若 X、Y 为向量，则必须满足 m=size(X)，n=

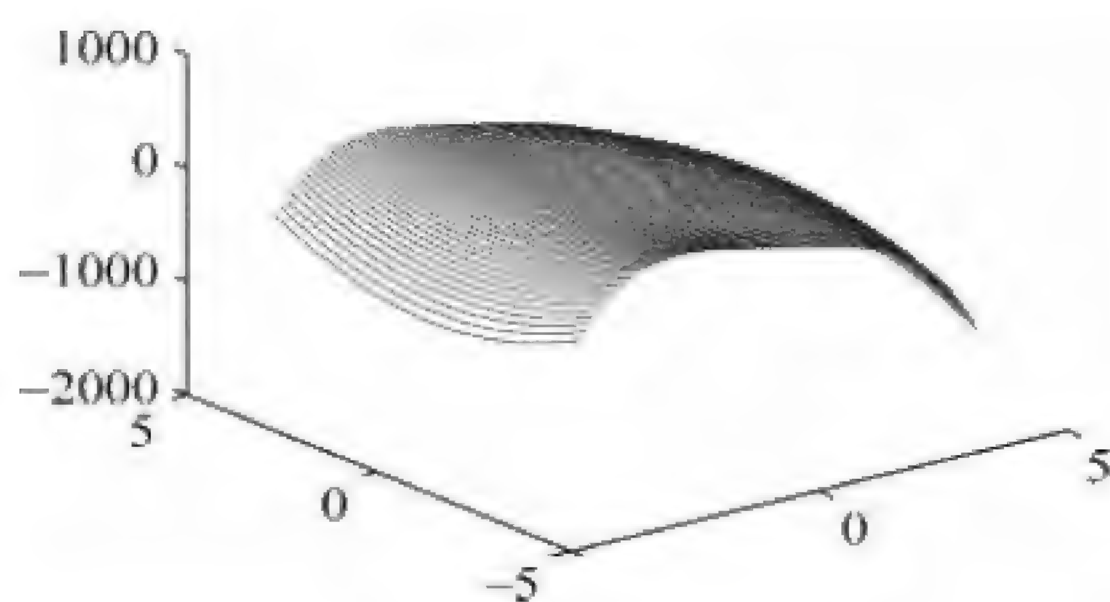


图 3-32　三维线条图形

size(Y),[m,n]=size(Z)。

surf(X,Y,Z,C)：以 Z 确定的曲面高度,C 确定的曲面颜色,按照 X、Y 形成的格点矩阵,创建一渐变的三维曲面。

surf(…,'PropertyName',PropertyValue)：设置曲面的属性。

surfc(…)：采用 surfc 函数的格式同 surf,同时在曲面下绘制曲面的等高线。

【例 3-25】 绘制球体的三维图形。

```
clear all
clc
figure
[X,Y,Z] = sphere(40);          % 计算球体的三维坐标
surf (X,Y,Z);                  % 绘制球体的三维图形
xlabel('x'),
ylabel('y'),
zlabel('z');
title(' shading faceted ');
```

输出图形如图 3-33 所示。

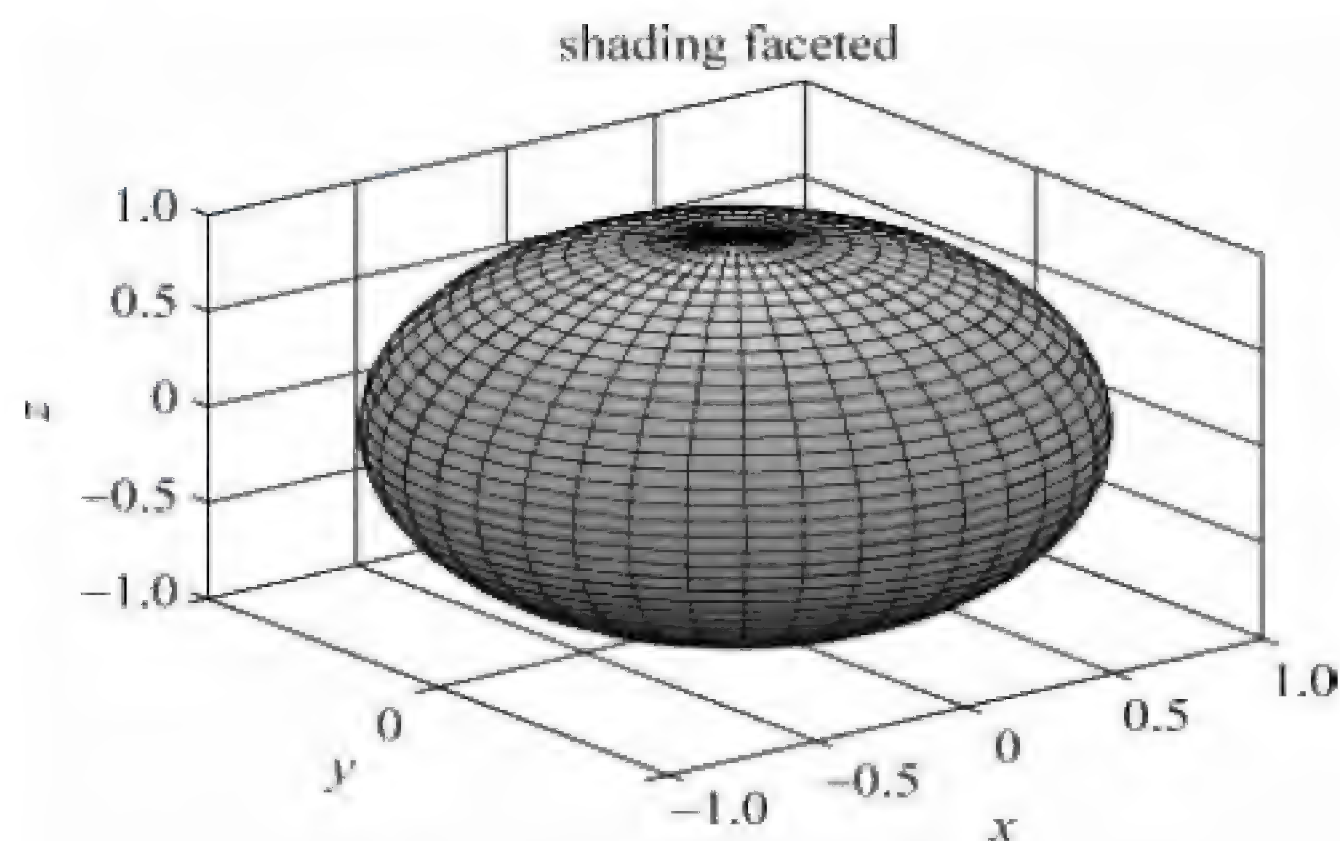


图 3-33　球体图形

注意：在图形窗口,需将图形的属性 Renderer 设置成 Painters,才能显示出坐标名称和图形标题。

在图 3-33 中,可以看到球面被网格线分割成小块,每一小块可看作是一块补片,嵌在线条之间。这些线条和渐变颜色可以由命令 shading 来指定,其格式如下：

shading faceted：在绘制曲面时采用分层网格线，为默认值。

shading flat：表示平滑式颜色分布方式，去掉黑色线条，补片保持单一颜色。

shading interp：表示插补式颜色分布方式，同样去掉线条，但补片以插值加色。这种方式需要比分块和平滑更多的计算量。

3.3.2 网格曲面隐藏线的显示和关闭

显示或不显示网格曲面的隐藏线将对图形的显示效果有一定影响。MATLAB 提供了相关的控制命令 hidden，调用这种命令的格式是 hidden on 或 hidden off。

hidden on：去掉网格曲面的隐藏线。

hidden off：显示网格曲面的隐藏线。

【例 3-26】 绘出有隐藏线和无隐藏线的函数 $f(x,y)=\frac{\cos(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$ 的网格曲面。

解：编写 MATLAB 代码如下：

```
clear all
clc
x = -7:0.4:7;
y = x;
[X,Y] = meshgrid(x,y);
R = sqrt(X.^2 + Y.^2) + eps;
Z = cos(R)./R;
subplot(1,2,1)
mesh(X,Y,Z)
hidden on
grid on
title('hidden on')
axis([-10 10 -10 10 -1 1])
subplot(1,2,2)
mesh(X,Y,Z)
hidden off
grid on
title('hidden off')
axis([-10 10 -10 10 -1 1])
```

运行上述代码后，得到如图 3-34 所示的图形。

3.3.3 三维绘图的实际应用

【例 3-27】 在一丘陵地带测量高度，x 和 y 方向每隔 50m 测一个点，得高度见表 3-2。试拟合一曲面，确定合适的模型，并由此找出最高点和该点的高度。

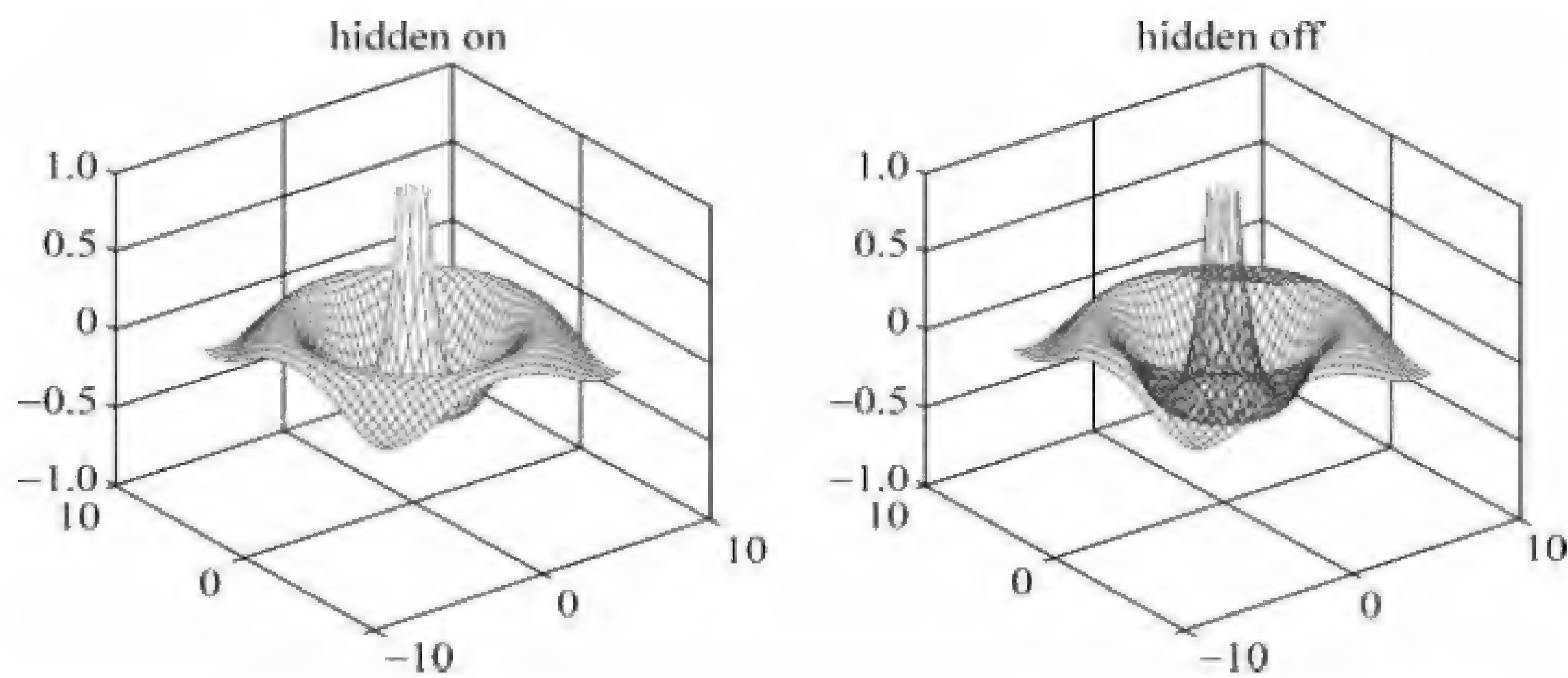


图 3-34 有无隐藏线的函数网格曲面图

表 3-2 高度数据

高度/m（y/m \ x/m）	100	200	300	400
100	536	597	524	278
200	598	612	530	378
300	580	574	498	312
400	562	526	452	234

解：在 MATLAB 中编写如下代码：

```
clear all
clc
x = [100 100 100 100 200 200 200 200 300 300 300 300 400 400 400 400];
y = [100 200 300 400 100 200 300 400 100 200 300 400 100 200 300 400];
z = [536 597 524 378 598 612 530 378 580 574 498 312 562 526 452 234];
xi = 100:10:400;
yi = 100:10:400;
[X,Y] = meshgrid(xi,yi);
H = griddata(x,y,z,X,Y,'cubic');
surf(X,Y,H);
view( - 112,26);
hold on;
maxh = vpa(max(max(H)),6)
[r,c] = find(H> = single(maxh));
stem3(X(r,c),Y(r,c),maxh,'fill')
title('高度曲面')
```

运行后得到高度曲面图像如图 3-35 所示。

同时得到最高点为

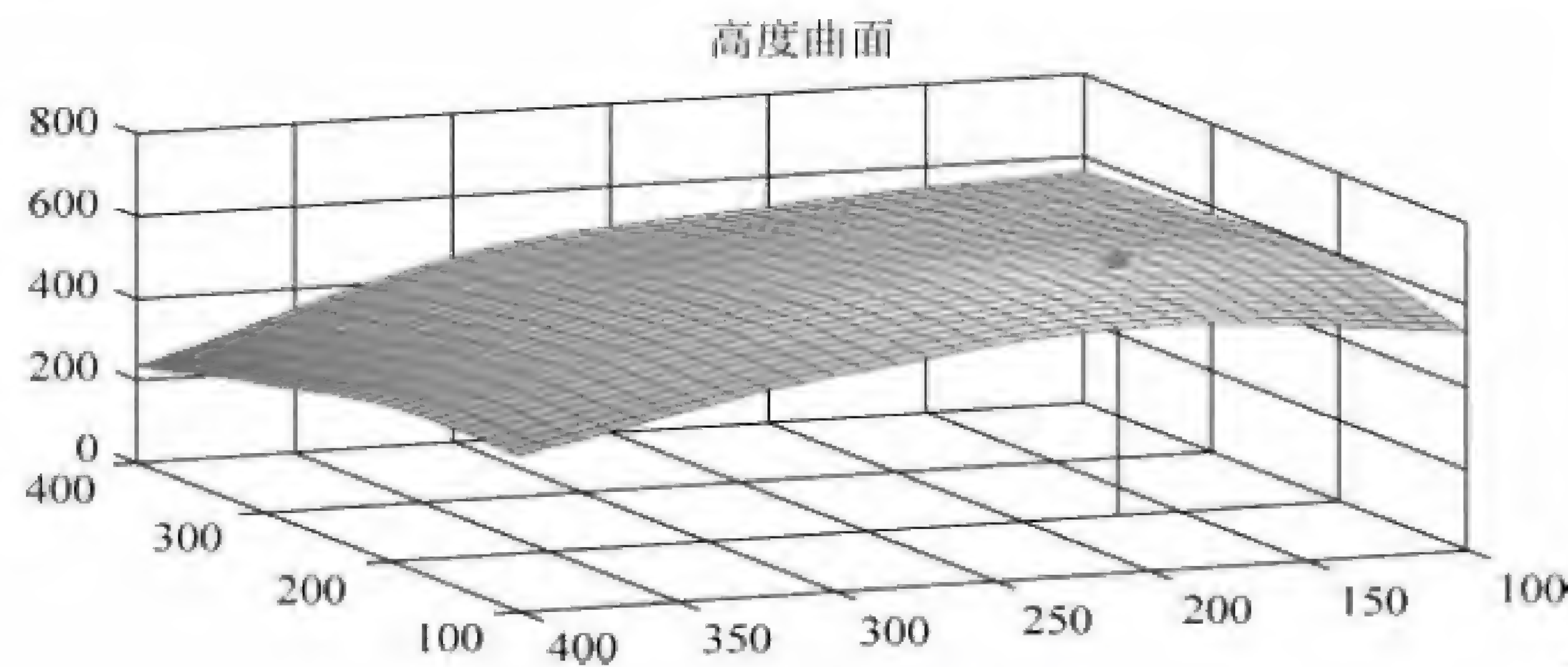


图 3-35　拟合的高度曲面

```
maxh =
616.113
```

即该丘陵地带高度最高点为 616.113m。

3.4　特殊图形的绘制

3.4.1　特殊二维图形的绘制

在 MATLAB 中，还有其他绘图函数可以绘制不同类型的二维图形，以满足不同的要求，表 3-3 列出了这些绘图函数。

表 3-3　其他绘图函数

函　　数	二维图的形状	备　　注
bar(x,y)	条形图	x 是横坐标，y 是纵坐标
fplot(y,[a b])	精确绘图	y 代表某个函数，[a b]表示需要精确绘图的范围
polar(θ,r)	极坐标图	θ 是角度，r 代表以 θ 为变量的函数
stairs(x,y)	阶梯图	x 是横坐标，y 是纵坐标
line([x1, y1],[x2,y2],…)	折线图	[x1, y1]表示折线上的点
fill(x,y,'b')	实心图	x 是横坐标，y 是纵坐标，'b'代表颜色
scatter(x,y,s,c)	散点图	s 是圆圈标记点的面积，c 是标记点颜色
pie(x)	饼图	x 为向量
contour(x)	等高线	x 为向量

【例 3-28】 用函数画一个条形图。

解：依据题意编写如下代码：

```
clear all
clc
x = -4:0.4:4;
bar(x,exp(-x.*x));
title('条形图')
```

输出图形如图 3-36 所示。

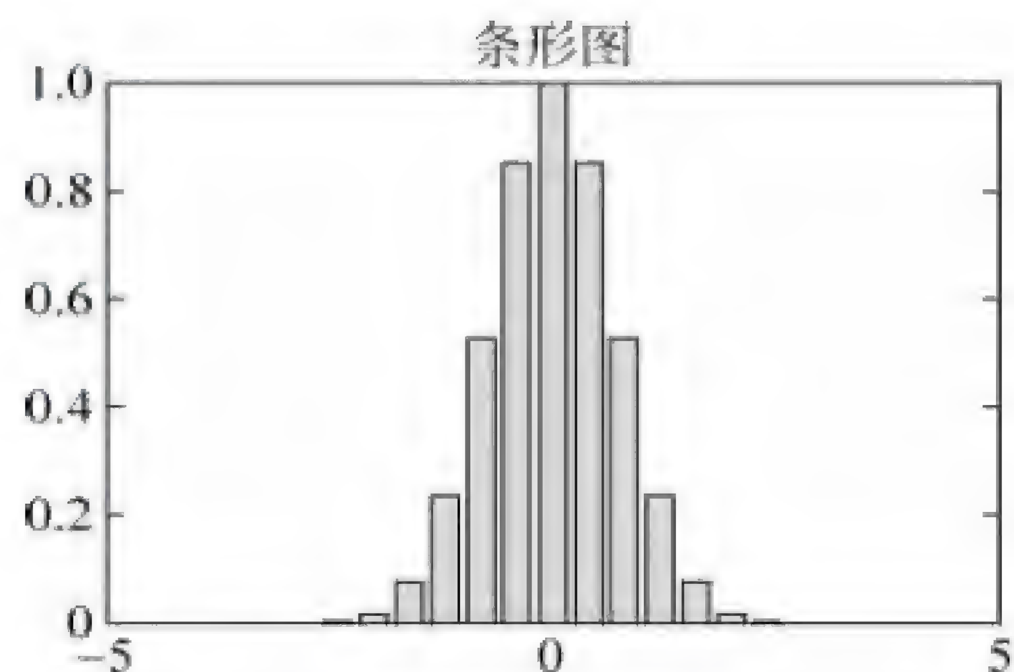


图 3-36　条形图

【例 3-29】 用函数画一个针状图。

解：依据题意编写如下代码：

```
clear all
clc
x = 0:0.05:4;
y = 2*(x.^0.3).*exp(-x);
stem(x,y)
title('针状图')
```

输出图形如图 3-37 所示。

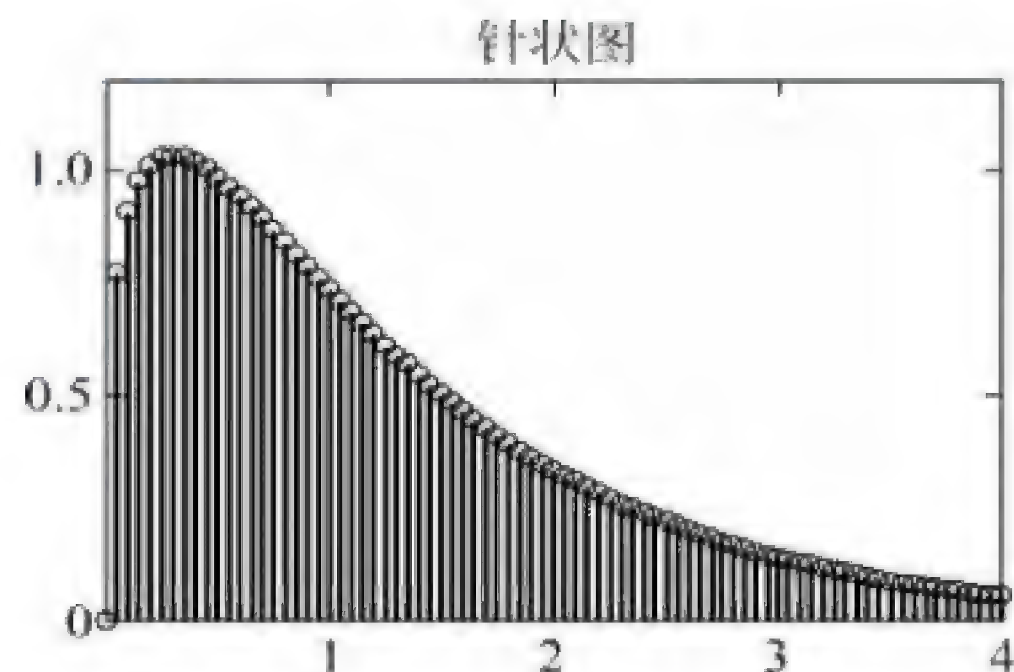


图 3-37　针状图

3.4.2　特殊三维图形

在科学研究中，有时也需要绘制一些特殊的三维图形，如统计学中的三维直方图、圆柱体图、饼状图等特殊样式的三维图形。

1. 螺旋线

在三维绘图中，螺旋线分为静态螺旋线、动态螺旋线和圆柱螺旋线。其中产生静态螺旋线的 MATLAB 代码如下：

```
clear all
clc
a = 0:0.2:10 * pi;
h = plot3(a. * cos(a),a. * sin(a),2. * a,'b','linewidth',2);
axis([ - 50,50, - 50,50,0,150]);
grid on
set(h,'erasemode','none','markersize',22);
title('静态螺旋线');
```

运行以上代码，得到如图 3-38 所示的图形。

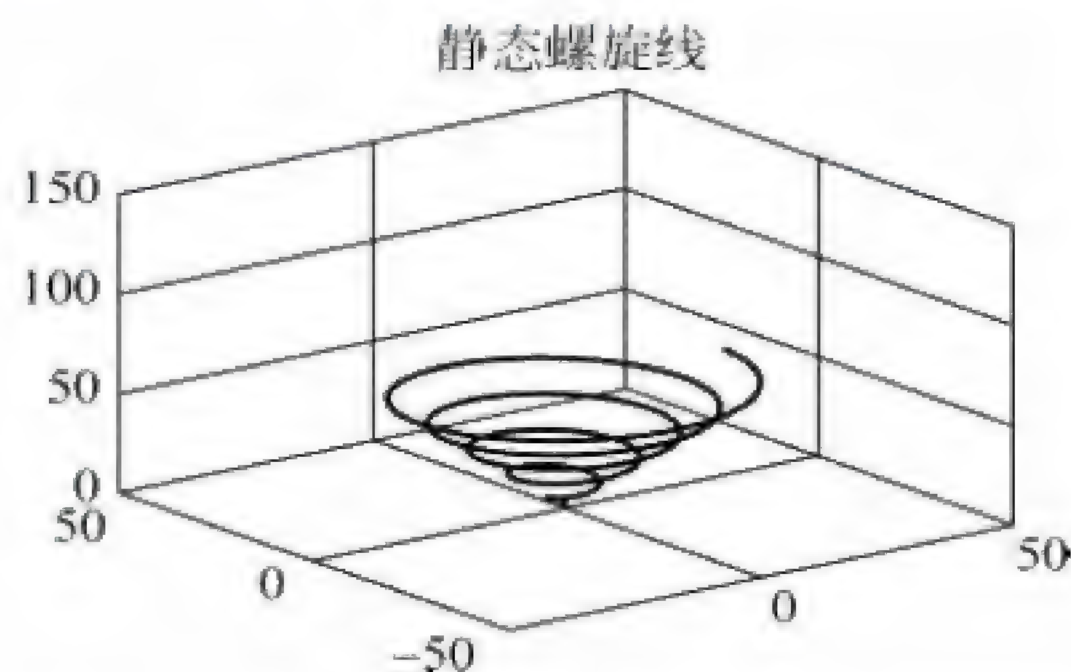


图 3-38　静态螺旋线图

产生动态螺旋线的 MATLAB 代码如下：

```
clear all
clc
t = 0:0.2:8 * pi;
i = 1;
h = plot3(sin(t(i)),cos(t(i)),t(i),' * ','erasemode','none');
grid on
axis([ - 1 1  - 1 1 0 30])
for i = 2:length(t)
    set(h,'xdata',sin(t(i)),'ydata',cos(t(i)),'zdata',t(i));
    drawnow
    pause(0.01)
end
title('动态螺旋线图像');
```

运行以上代码，得到如图 3-39 所示的图形。

2. 柱状图

与二维情况相类似，MATLAB 提供了两类画三维直方图的命令：一类是用于画垂直放置的三维直方图；另一类是用于画水平放置的三维直方图。

1）垂直放置的三维直方图

MATLAB 中绘制垂直放置的三维直方图函数格式如下：

bar3(Z)：以 $x=1,2,3,\cdots,m$ 为各个数据点的 x 坐标，以 $y=1,2,3,\cdots,n$ 为各个数据点的 y 坐标，以 Z 矩阵的各个对应元素为 z 坐标（Z 矩阵的维数为 $m\times n$）；

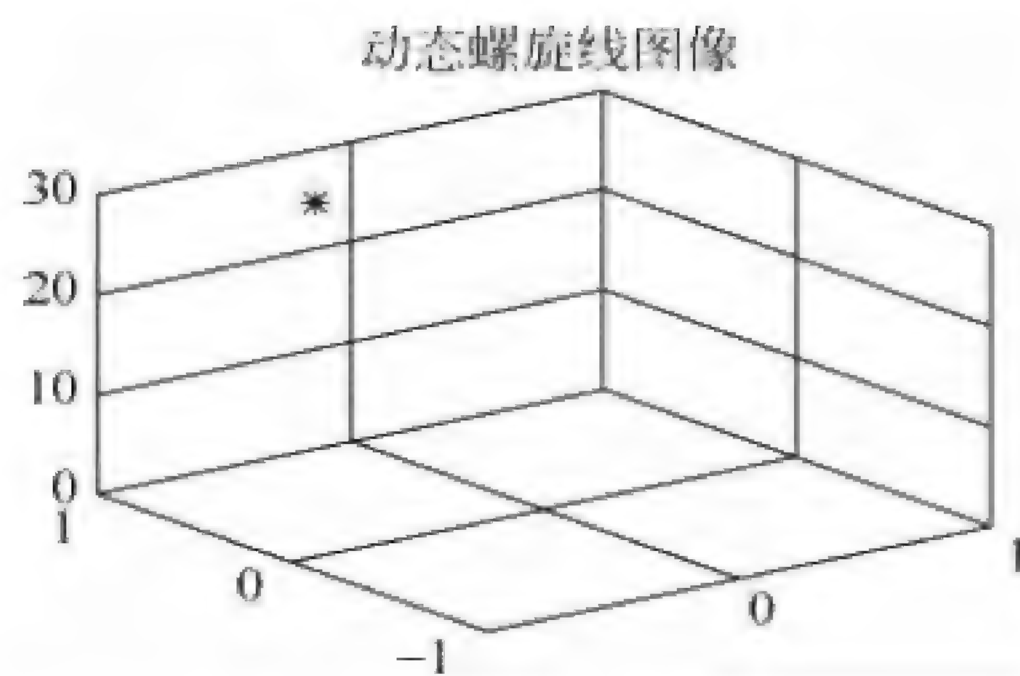


图 3-39　动态螺旋线

bar3(Y,Z)：以 $x=1,2,3,\cdots,m$ 为各个数据点的 x 坐标，以 Y 向量的各个元素为各个数据点的 y 坐标，以 Z 矩阵的各个对应元素为 z 坐标(Z 矩阵的维数为 $m\times n$)；

bar3(Z,option)：以 $x=1,2,3,\cdots,m$ 为各个数据点的 x 坐标，以 $y=1,2,3,\cdots,n$ 为各个数据点的 y 坐标，以 Z 矩阵的各个对应元素为 z 坐标(Z 矩阵的维数为 $m\times n$)，并且各个方块的放置位置由字符串参数 option 来指定(detached 为分离式三维直方图；grouped 为分组式三维直方图；stached 为累加式三维直方图)。

2）水平放置的三维直方图

MATLAB 中绘制水平放置的三维直方图的函数包括 bar3h(Z)、bar3h(Y,Z)、bar3h(Z,option)。它们的功能及使用方法与上述的 3 个 bar3 命令的功能及使用方法相同。

【例 3-30】 利用函数绘制出不同类型的直方图。

解：编写以下 MATLAB 代码：

```
clear all
clc
Z = [15,35,10;20,10,30]
subplot(2,2,1)
h1 = bar3(Z,'detached')
set(h1,'FaceColor','W')
title('分离式直方图')
subplot(2,2,2)
h2 = bar3(Z,'grouped')
set(h2,'FaceColor','W')
title('分组式直方图')
subplot(2,2,3)
h3 = bar3(Z,'stacked')
set(h3,'FaceColor','W')
title('叠加式直方图')
subplot(2,2,4)
h4 = bar3h(Z)
set(h4,'FaceColor','W')
```

运行以上代码，得到如图 3-40 所示的效果图。

3. 三维等高线

MATLAB 中提供的三维等高线的绘制函数格式如下：

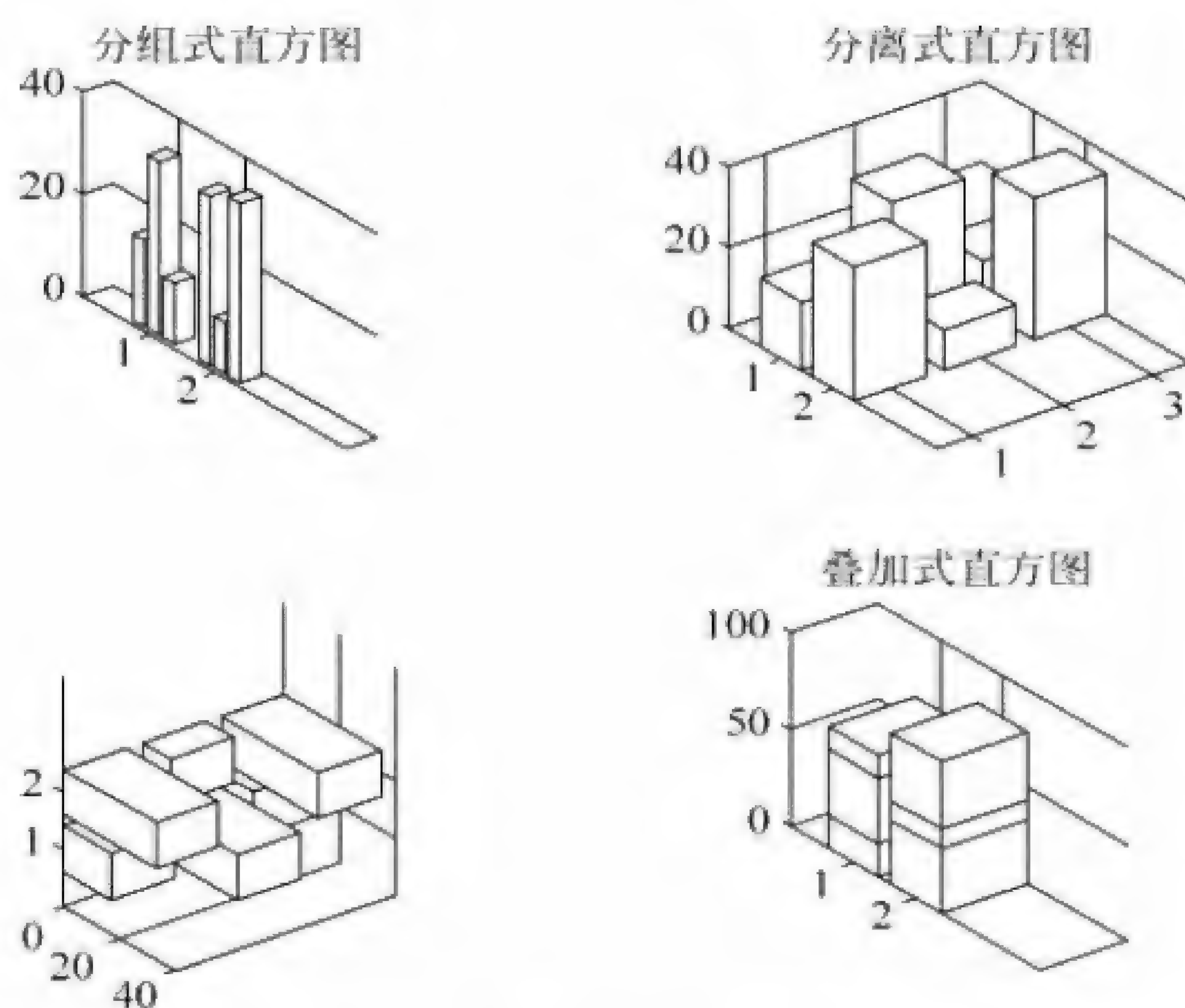


图 3-40　不同类型的三维直方图

contour3(X,Y,Z,n,option)：参数 n 指定要绘制出 n 条等高线，若默认参数 n，则系统自动确定绘制等高线的条数；参数 option 指定了等高线的线型和颜色。

clabel(c,h)：标记等高线的数值，参数 c、h 必须是 contour 命令的返回值。

【例 3-31】　绘制下列函数的曲面及其对应的三维等高线：

$$f(x,y)=2(1-x)^2\mathrm{e}^{-x^2-(y+1)^2}-8\left(\frac{x}{6}-x^3-y^5\right)\mathrm{e}^{-(x^2-y^2)}-\frac{1}{4}\mathrm{e}^{-(x+1)^2-y^2}$$

解：编写 MATLAB 代码如下：

```
clear all
clc
x = - 4:0.3:4;
y = x;
[X,Y] = meshgrid(x,y);
Z = 2 * (1 - X).^2. * exp( - (X.^2) - (Y + 1).^2)...
  - 8 * (X/6 - X.^3 - Y.^5). * exp( - X.^2 - Y.^2)...
  - 1/4 * exp( - (X + 1).^2 - Y.^2);
subplot(2,2,1)
mesh(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('Z')
title('Peaks 函数图形')
subplot(2,1,2)
[c,h] = contour3(x,y,Z);
clabel(c,h)
xlabel('x')
ylabel('y')
zlabel('z')
title('Peaks 函数的三维等高线')
```

运行代码后，得到如图 3-41 所示结果。

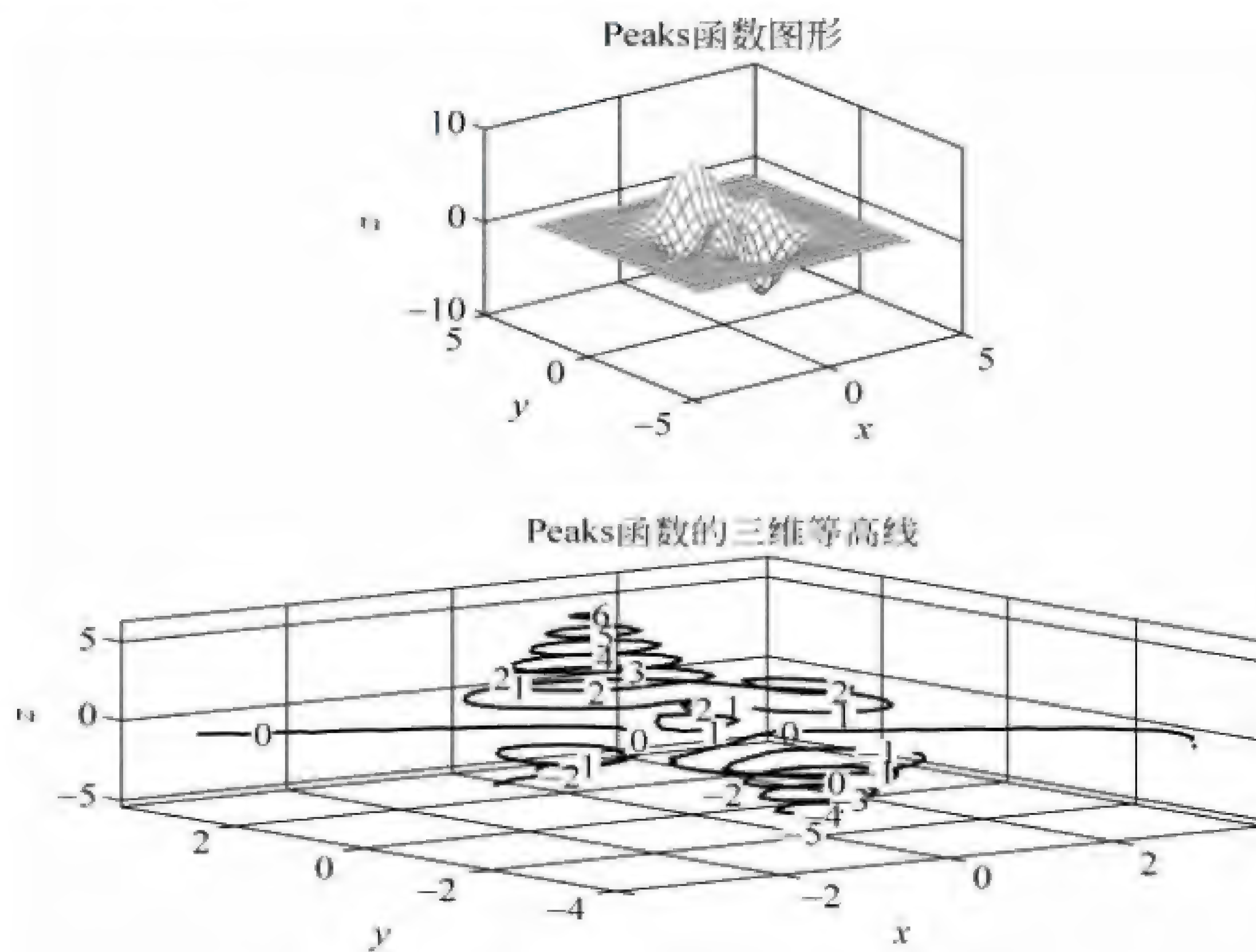


图 3-41 函数曲面及其对应的三维等高线

本章小结

本章首先介绍了数据图像的绘制，然后重点介绍了 MATLAB 二维绘图和三维绘图的知识，最后对 MATLAB 中的一些特殊图形绘制做了简单介绍。通过本章的学习，能让读者掌握 MATLAB 的各种基础绘图方法。

第4章 GUI应用

GUI(graphical user interfaces,图形用户界面)是由窗口、文字说明、菜单等对象构成的一个用户界面。用户可以通过一定的方法选择、激活这些图形对象,使计算机产生某种动作或变化,如实现计算、绘图等功能。

本章主要介绍了GUI设计的应用知识。

学习目标:

(1) 了解GUI的基础概念;

(2) 了解GUIDE的使用;

(3) 了解使用M文件创建GUI对象。

4.1 GUI基础概念

图形用户界面(graphical user interface)简称GUI,又称图形用户接口,是指采用图形方式显示的计算机操作用户界面。

与早期计算机使用的命令行界面相比,图形界面对于用户来说在视觉上更易于接受。然而这种界面若要通过显示屏的特定位置,并且以美观而不单调的视觉消息提示用户状态的改变,势必比简单的消息呈现花费更多的计算时间。

4.1.1 GUI开发方法

在进行某种方法的演示、制作一个可重复使用且操作简单的专用工具时,GUI是最好的选择。提供GUI的应用程序能够使用户的学习和使用变得方便和容易。用户不需要知道应用程序究竟是怎样执行各种命令的,只需要了解界面组件的使用方法,通过与界面交互就可以使指定的行为正确执行。

在MATLAB中,GUI是一种包含多个对象的图形窗口。用户必须对每一个对象进行界面布局和编程,从而使用户激活GUI每个对象时都能够执行设置好的行为。另外,用户必须保存和发布所创建的GUI,使GUI能真正地得到运行。

MATLAB为用户开发GUI界面提供了一个方便高效的集成环境——GUIDE（MATLAB图形用户界面开发环境）。GUIDE是一个用来界面设计的工具集。

MATLAB将所有GUI支持的用户控件都集成在这个环境中并提供界面外观等属性的设置方法。GUIDE将用户设计好的GUI界面保存在一个FIG文件中，同时还能够生成包含GUI初始化和组件布局控制代码的M文件。

1. FIG文件

FIG文件包含GUI图形窗口及其所有后裔的完全描述，包括所有对象的属性值。FIG文件是一个二进制文件，调用hgsave命令和界面设计编辑器的File菜单中的Save选项保存图形窗口时，将产生该文件。

FIG文件包含序列化的图形窗口对象。在用户打开GUI时，MATLAB能通过读取FIG文件重新构造图形窗口及其所有后裔。所有对象的属性都被设置为图形窗口创建时保存的属性。

FIG文件最大的功能是保存和引用对象句柄。可以使用open、openfig和hgload命令来打开一个后缀为.fig的文件。

2. M文件

该文件包括GUI设计、控制函数以及定义为子函数的用户控件回调函数，主要用于控制GUI展开时的各种特征。

应用程序M文件使用openfig命令来显示GUI。

提示：M文件不包含GUI的任何代码，这些代码完全由FIG文件保存。

GUIDE可以根据用户GUI的版本设计过程直接自动生成M文件框架，这样就简化了GUI的创建工作，用户可以直接使用这个框架来编写自己需要的函数代码。

实现一个GUI主要包括两个工作：GUI界面设计和GUI组件编程。整个GUI的实现过程可以分为以下几步：

（1）通过设置GUIDE应用程序选项来进行GUIDE组态；

（2）使用GUI编辑器来进行GUI界面设计；

（3）掌握M文件中所使用的编辑计算；

（4）编写GUI组件行为响应控制（即回调函数）代码。

4.1.2 GUI基本元素

GUIDE包含所有能够在GUI中使用的用户界面控件。这些控件都属于MATLAB的用户对象，可以通过Callback属性来进行回调函数的编程。

创建MATLAB的GUI必须具有以下三种基本元素：

1. 图形窗口

GUI每一个组件都必须安排在图形窗口，在画数据图像时，图形窗口通常会被自动创建。但还可以用函数figure来创建空图形窗口，空图形窗口经常用于放置各种类型的

组件。

2. 组件

在 MATLAB 的 GUI 中，每一个项目都是一个图形化组件。组件包括图形化控件（按钮、编辑框等）、静态元素（窗口、文本字符串）、菜单和坐标系。

图形化控件和静态元素由函数 uicontrol 创建，菜单由函数 uimenu 和 uicontextmune 创建，坐标系经常用于显示图形化数据，由 axes 创建。

3. 回应

如果用户用鼠标单击或用键盘输入一些信息时，那么程序就要有相应的动作。鼠标单击或输入信息是一个事件，如果 MATLAB 程序运行相应的函数，那么 MATLAB 函数肯定会有所反应。

例如，如果用户单击某一按钮，这个事件必然导致相应的 MATLAB 语句被执行。这些相应的语句被称为回应。只要执行了 GUI 的单个图形组件，就必须有一个回应。

下面简单介绍几种控件的概念和特点。

(1) 按钮：通过单击按钮可以实现某种行为并调用相应的回调子函数。

(2) 拴牢按钮：产生一个二进制状态的行为。单击该按钮将使按钮的外观保持陷下的状态，同时调用相应的回调函数。再次单击后，该按钮在弹起的同时也调用某一回调函数。拴牢按钮的回调函数首先要对按钮的状态进行查询，然后才能决定相应的行为。

(3) 单选按钮：单选按钮与按钮的执行方式没有本质的区别，但是单选按钮通常以组为单位，一组单选按钮之间是一种互相排斥的关系。

(4) 复选框：复选框与单选按钮类似，只是多个复选框可以同时有效。复选框为用户提供一些可以独立选择的选项进行设置程序模式，例如是否显示工具条、是否生成回调函数原型等。

(5) 编辑框：编辑框是控制用户编辑或修改字符串的文本区，其属性包含用户输入的文本信息。

(6) 静态文本：静态文本通常作为其他控件的标签使用，用户不能采用交互方式修改静态文本或调用相应的回调函数。

(7) 滚动条：使用户能够通过移动条来改变指定范围内的数值输入，滚动条的位置代表用户输入的数值。

(8) 组合框：组合框是图形窗口的一个封闭区域，它把相关联的控件组合在一起，使用户界面更容易理解。

(9) 列表框：显示由属性定义的一系列列表项并使用户能够选择其中的一项或多项。GUI 中可以存在任意个 GUI 组件，使用时需要保证各个组件的名称和属性有所区别。

4.1.3 GUI 的层次

实现一个 GUI 的过程必须完成两个基本任务：第一个是组建布局，第二个是组建的

编程。如果用户需要将开发的GUI得到真正应用,还需要保持并发布其所开发的GUI,这些操作都是在GUI的环境下进行的。

在MATLAB中,一幅图可以拥有多个图形对象,且每个图形对象都可以被单独地操作。在MATLAB中,每个图形对象都有一个句柄。如果要对图形对象进行控制或者修改图形对象的属性,就需要获得这些句柄。

在进行GUI设计的时候,需要准确理解图形对象及其句柄。图形对象不仅包括uimenu、uicontrol和uicontextmenu对象,还包括图形、坐标轴、线条、曲面、文本和它们的子对象。

GUI对象层次结构如图4-1所示。

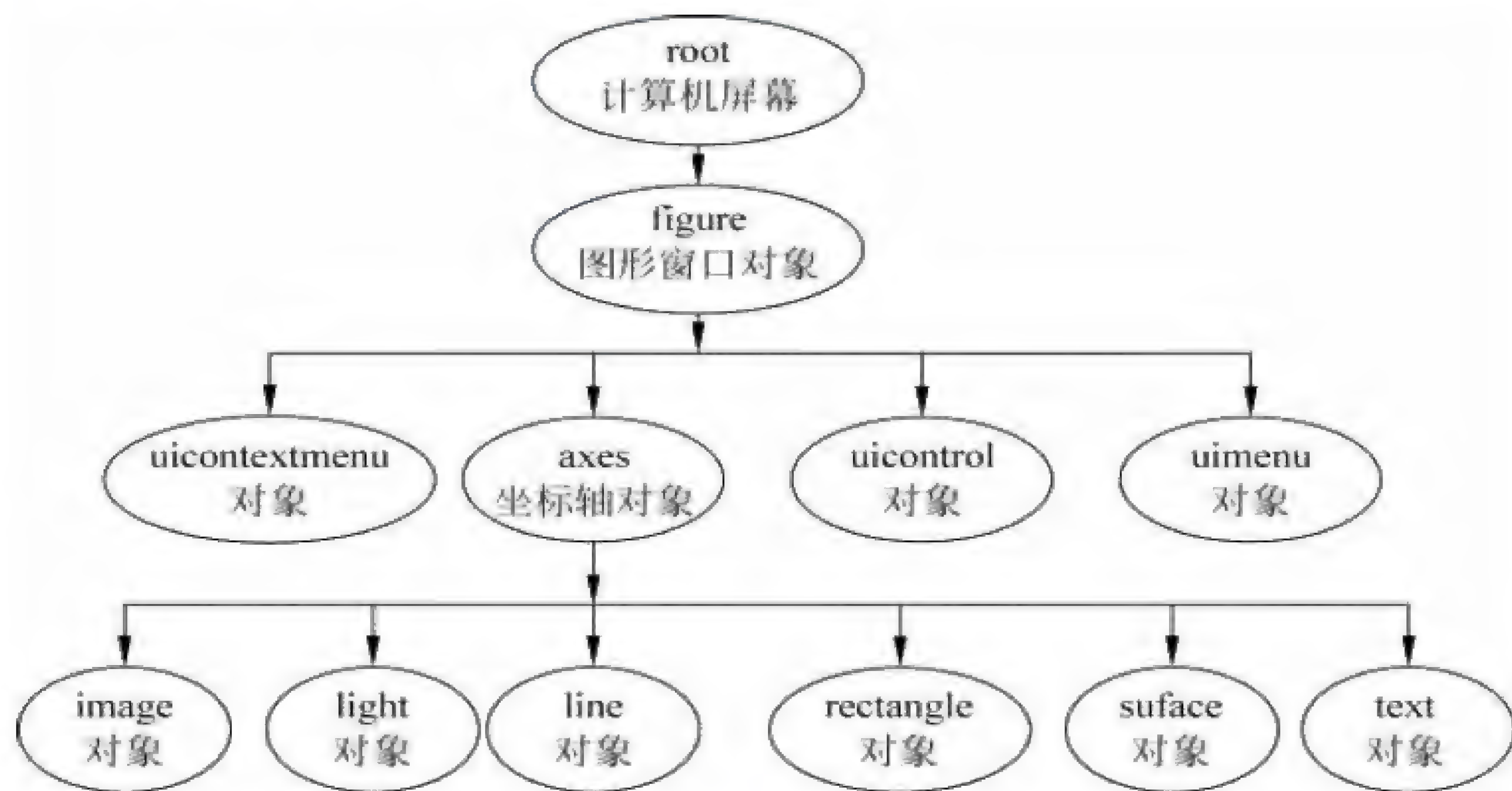


图4-1 GUI对象层次结构

图中root对象是根对象。它在GUI对象层次结构图的最上层,是由系统在启动MATLAB时自动创建的。每个图形窗口都可以包含多个对象(如uimenu对象、uicontrol对象等),这些对象也可以包含多个对象。uicontrol对象是一个无子对象的结点,但它有很多种类型,如文本框、编辑框等。

其他的所有对象都是坐标轴的子对象,它们在坐标轴上显示。当利用MATLAB函数来创建子对象时,如果父对象暂时不存在,那么创建对象的函数会自动为它们建立父对象。

句柄图形是对底层图形对象集合的总称,它进行了生成图形的工作。通过句柄,用户可以设置图形对象的许多属性。每个图形对象句柄都由一个唯一的数字来标识,这个唯一的标识叫做句柄。每创建一个新的对象,系统就会为它建立一个唯一的句柄,用来唯一地识别这个对象,这和VC编程中的句柄概念是相同的。

4.2 菜单

菜单是用户图形界面的一个重要组成部分。一般来说,菜单位于图形窗口的最上面,且通常在顶层菜单下构成下拉子菜单,用户可以通过单击鼠标选择下拉子菜单中的

单项，得到所需要的菜单功能。

一个菜单项可以用菜单项列表作为子菜单，而这个子菜单还可以继续嵌套带有其他功能的子菜单，但菜单层次的数目会受到窗口系统的限制。

4.2.1 建立菜单和子菜单

在 MATLAB 中建立菜单可以用 uimenu 函数，该函数不仅用于建立一级菜单项，也可以建立多级子菜单项。

建立一级菜单项的函数调用格式为

```
handle = uimenu('PropertyName',PropertyValue,…)
```

一级菜单项句柄＝uimenu(图形窗口句柄，属性名 1，属性值 1，属性名 2，属性值 2，…)

建立子菜单项的函数调用格式为

```
handlesub = uimenu(parent,'PropertyName',PropertyValue,…)
```

子菜单项句柄＝uimenu(一级菜单项句柄，属性名 1，属性值 1，属性名 2，属性值 2，…)

uimenu 对象中最重要的属性是 Label 和 Callback。其中，Label 属性值是菜单条和下拉菜单项上的文本字符串，用来确认菜单项。Callback 属性值是 MATLAB 字符串。

注意：建立一级菜单项时，要给出图形窗口句柄。否则，就会在当前窗口中建立菜单项。如果没有活动窗口，则会自动打开一个图形窗口。

建立子菜单项时，必须指定一级菜单项对应的句柄值。

【例 4-1】 用 uimenu 函数创建图形窗口中的菜单，并添加几个子菜单。

解：首先建立一个顶部菜单，并将其命名为 GUI4_1，在命令行窗口中输入：

```
handle = uimenu(gcf,'Label','GUI4_1');
```

运行以上语句得到结果如图 4-2 所示。从图中可以看出，MATLAB 默认把 GUI4_1 放在最后。

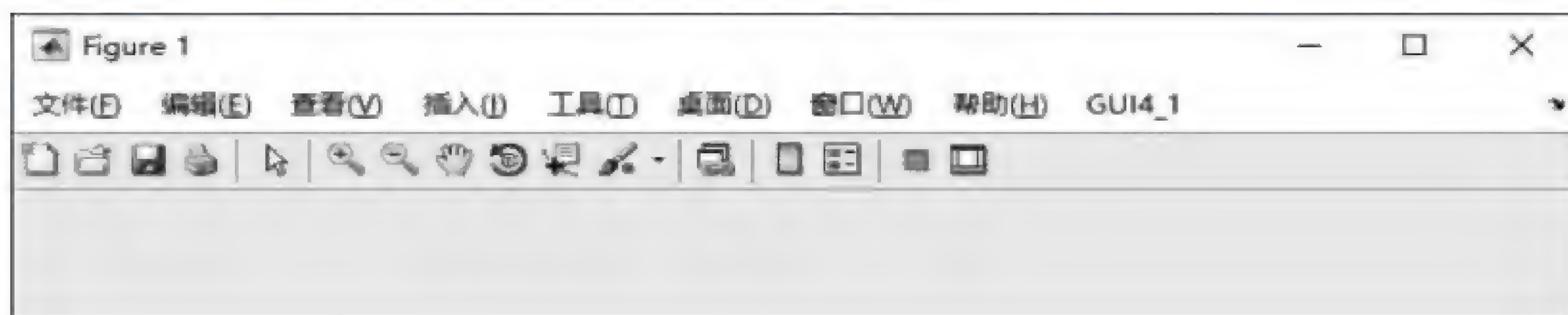


图 4-2 建立的菜单图形

然后，在图 4-2 建立的菜单下新建立三个子菜单项，分别命名为 Hold、Fun、Grid on。其中，Hold 可以保持前一步操作的结果，Fun 可以选择不同函数的视图，Grid on 可以用于切换坐标轴中格栅的状态。

建立具有保持功能的子菜单 Hold，在 MATLAB 命令行窗口中输入：

```
handlesub1 = uimenu(handle,'Label','Hold','Callback','hold');
```

运行以上语句得到结果如图 4-3 所示。从图中可以看出，菜单 GUI4_1 下方出现了其子菜单 Hold。

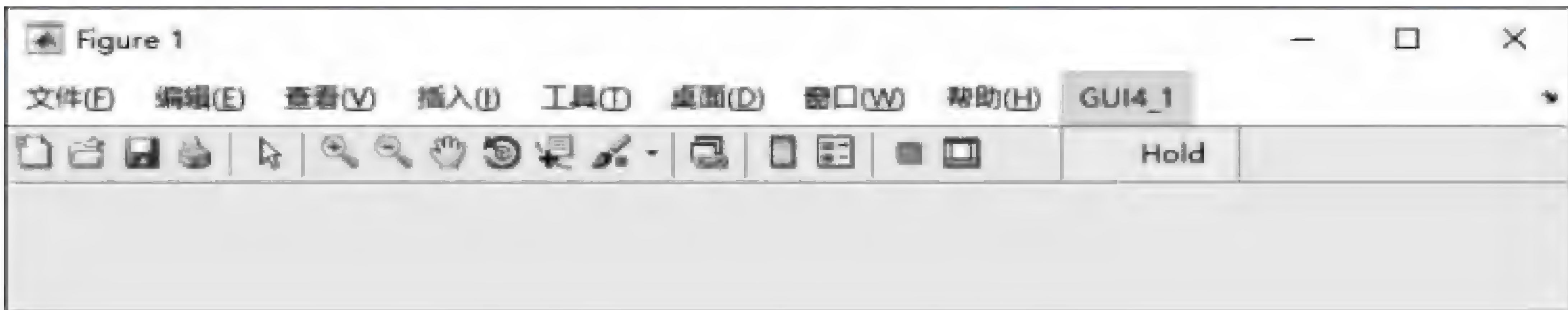


图 4-3　建立子菜单 Hold 的图形

建立子菜单 Fun，在 MATLAB 命令行窗口中输入：

```
handlesub2 = uimenu(handle,'Label',' Fun');
```

建立子菜单 Fun 的子菜单：正切函数 tan，在 MATLAB 命令行窗口中输入：

```
handlesub21 = uimenu(handlesub2,'Label','tan', …
                    'Callback','plot(tan([(-pi/2)+0.01:0.01:(pi/2)-0.01]),''r'')');
```

建立子菜单 Fun 的子菜单：余切函数 cot，在 MATLAB 命令行窗口中输入：

```
handlesub22 = uimenu(handlesub2,'Label','cot', …
                    'Callback','plot(cot([0.01:0.01:pi-0.01]),''b:'')');
```

运行以上语句得到结果如图 4-4 所示。从图中可以看出，菜单 GUI4_1 下方出现了其子菜单 Fun 和 Fun 的子菜单函数(tan 和 cot)。



图 4-4　建立子菜单 Fun 的子菜单图形

建立 Grid 子菜单，在 MATLAB 命令行窗口中输入：

```
handlesub3 = uimenu(handle,'Label','Grid on','Callback','grid');
```

运行以上语句得到结果如图 4-5 所示。从图中可以看出，菜单 GUI4_1 下方出现了其子菜单 Grid on。

在图 4-5 中，依次执行 Hold、tan、cot、Grid on 后，得到如图 4-6 所示的结果，其中 Hold 的功能就是把正切函数和余切函数的图形同时显示在一个图形窗口上，Grid on 用

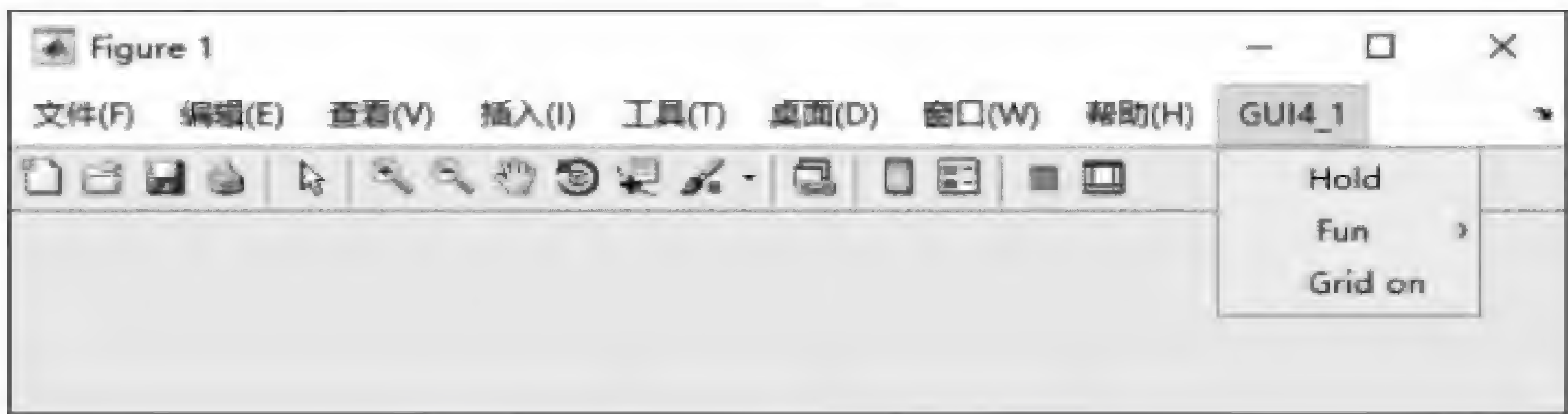


图 4-5 建立子菜单 Grid on 图形

于显示图形坐标轴的格栅。

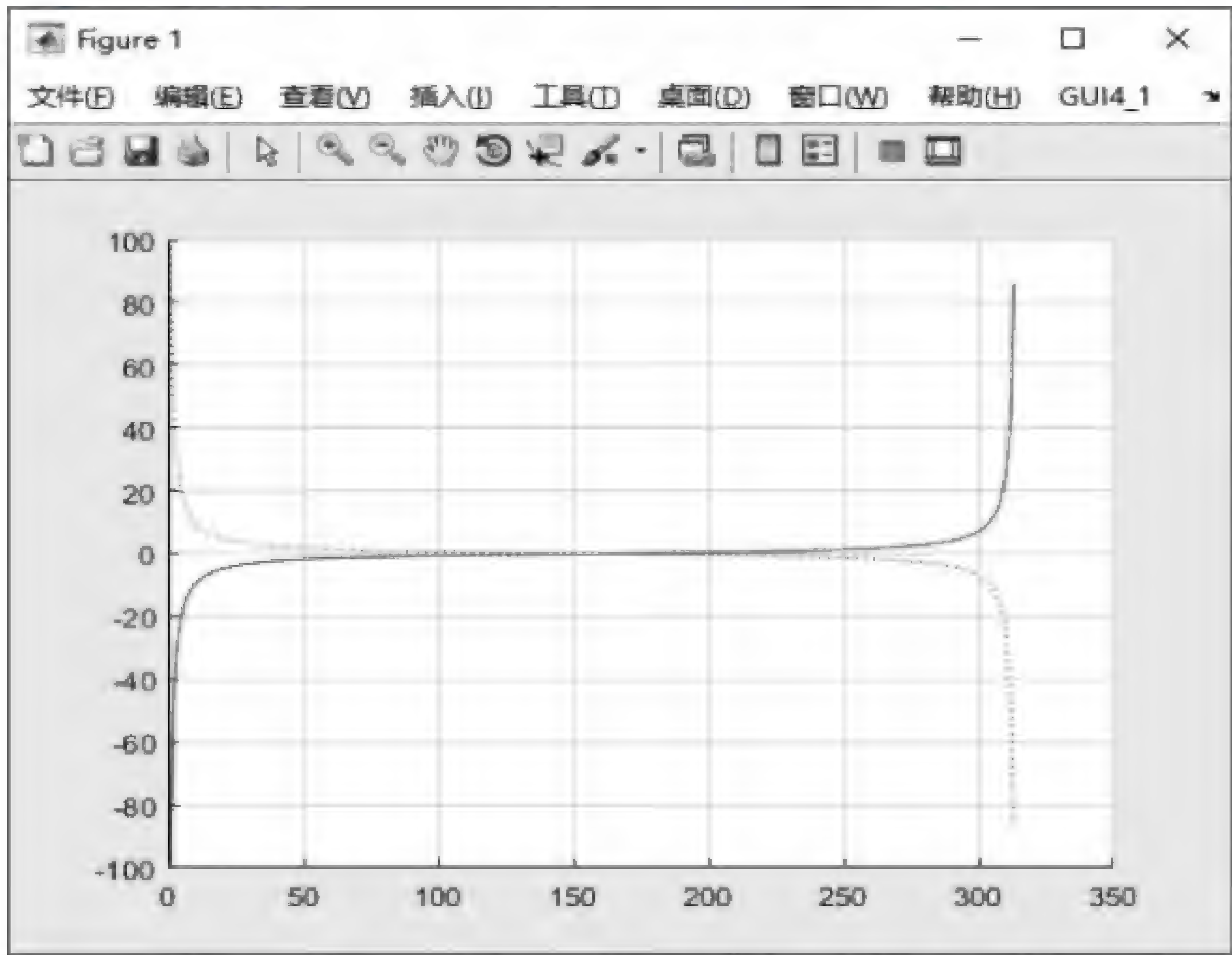


图 4-6 运行子菜单选项得到的结果图形

最后，再建立一个顶部菜单 Shut，用于关闭前面几步中建立的菜单以及关闭整个图形窗口。

建立顶部菜单 Shut，在 MATLAB 命令行窗口中输入：

```
handleh = uimenu(gcf,'Label','Shut');
```

建立具有关闭前面几步中建立的菜单功能的子菜单，在 MATLAB 命令行窗口中输入：

```
handlesub1 = uimenu(handleh,'Label','Remove',...
             'Callback','delete(handle);drawnow');
```

建立具有关闭整个图形窗口功能的子菜单，在 MATLAB 命令行窗口中输入：

```
handlehsub2 = uimenu(handleh,'Label','shut Figure','Callback','close');
```

运行以上程序,得到结果如图 4-7 所示。

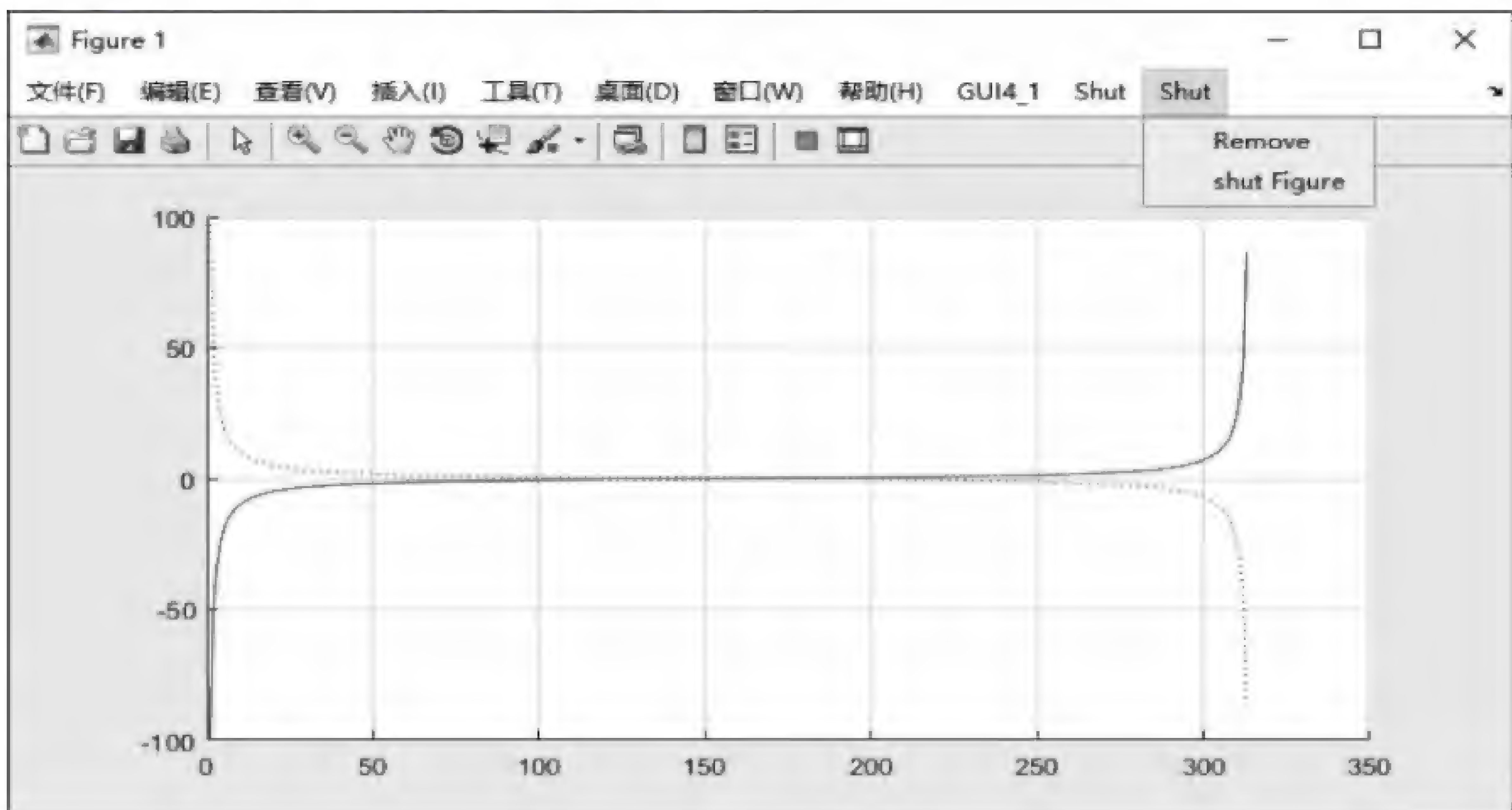


图 4-7　新建立的顶部菜单 Shut

当运行图 4-7 中顶部菜单 Shut 下的 Remove 子菜单后,题中前面建立的菜单项就会关闭,如图 4-8 所示。

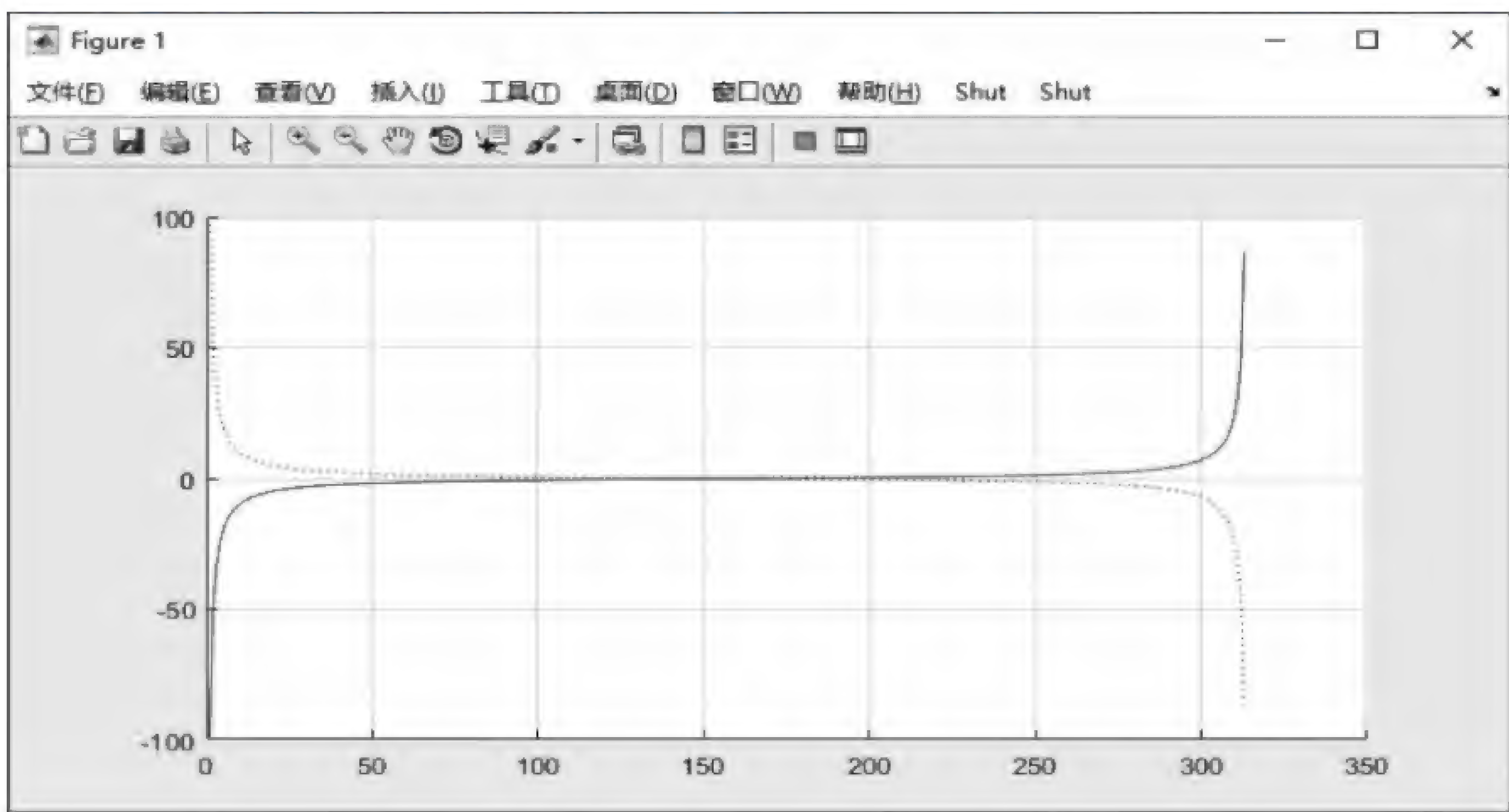


图 4-8　关闭 GUI4_1 菜单的效果图

如果运行顶部菜单 Shut 下的 shut Figure 子菜单,就可以关闭整个图形窗口。

4.2.2 菜单对象常用属性

菜单对象的属性值不但可以定义对象的性质，还能控制菜单如何显示，决定选择菜单项所引起的动作，所以熟悉菜单对象属性是十分必要的。本节重点介绍了菜单对象的几个常用属性。

1. Name 属性

Name 属性的取值可以是任何字符串，它的默认值为空。这个字符串作为图形窗口的标题。一般情况下，其标题形式为 Figure No. 1：字符串。

2. MenuBar 属性

MenuBar 属性的取值可以是 figure(默认值)或 none，用来控制图形窗口是否具有菜单条。如果它的属性值为 none，则表示该图形窗口没有菜单条，这时用户可以根据后面将要介绍的 uimenu()函数来加入自己的菜单条。如果属性值为 figure，则该窗口将保持图形窗口默认的菜单条，这时也可以采用 uimenu()函数在原来默认的图形窗口菜单后面添加新的菜单项。

3. NumberTitle 属性

NumberTitle 属性的取值是 on(默认值)或 off，决定着在图形窗口的标题中是否以"Figure No. n："为标题前缀，这里 n 是图形窗口的序号，即句柄值。

4. Type 属性

Type 属性的取值总是 uimenu，这个属性值表明图形对象的类型。对菜单对象，其类型就是 uimenu，用户不能改写这个属性。

5. Tag 属性

Tag 属性的取值是字符串，它定义了该菜单对象的一个标识值。定义了 Tag 属性后，在任何程序中都可以通过这个标识值找出该菜单对象。

6. UserData 属性

UserData 属性的取值是一个矩阵，默认值为空矩阵，用户可以在这个属性中保存与该菜单对象相关的重要数据或信息，达到传递数据或信息的目的，并可以用 set 和 get 函数访问该属性。

菜单对象除具有 Children、Parent、Tag、UserData、Visible 等公共属性外，还有一些常用的特殊属性，如表 4-1 所示。

表 4-1 一些常用的特殊属性

属性名	属性值及作用
Label	取值字符串，用于定义菜单项的名字。可以在字符串中加“&”——对应于下画线，可用 Alt 激活
Accelerator	取值任何字母，用于定义菜单的快捷键
Callback	取值字符串，可以是某个 M 文件的文件名或一组 Matlab 命令。该菜单被选中后，自动调用此回调函数
Checked	取值 on 或 off，为菜单项定义一个标记，指明菜单项是否被选中
Enable	取值 on 或 off，控制菜单项的可选择性。不可用时，该菜单呈现灰色
Position	定义一级菜单在菜单栏上的相对位置或子菜单项在菜单组内的相对位置。默认为1——最左端
Separator	取值为 on 或 off。可以用分隔线将各菜单项分开

4.2.3 快捷菜单

快捷菜单是用鼠标右击某对象时在屏幕上弹出的菜单。这种菜单出现的位置是不固定的，而且总是和某个图形对象相联系。在 MATLAB 中，可以使用 Accelerator 函数或 & 符号来建立快捷菜单。

【例 4-2】 将例 4-1 中建立的 Shut 菜单项及其下拉菜单 Remove 子菜单各带一个快捷键，而另一项下拉子菜单 shut Figure 带另一个快捷键。

解：在 MATLAB 命令行窗口中输入：

```
handle = uimenu(gcf,'Label','&Shut');                    % 带快捷键 S 的用户菜单 Shut
handlesub1 = uimenu(handle,'Label','&Remove',...         % 带快捷键 R 的下拉菜单 Remove
    'Callback','set(gcf,''shut'',''remove'')');
handlesub2 = uimenu(handle,'label','shut Figure',..      % 制作另一个下拉菜单 shut Figure
    'Callback','set(gcf,''Shut'',''shut Figure'')',...
    'Accelerator','f');                                  % 为 shut Figure 菜单设置快捷键 F
```

运行结果如图 4-9 所示。

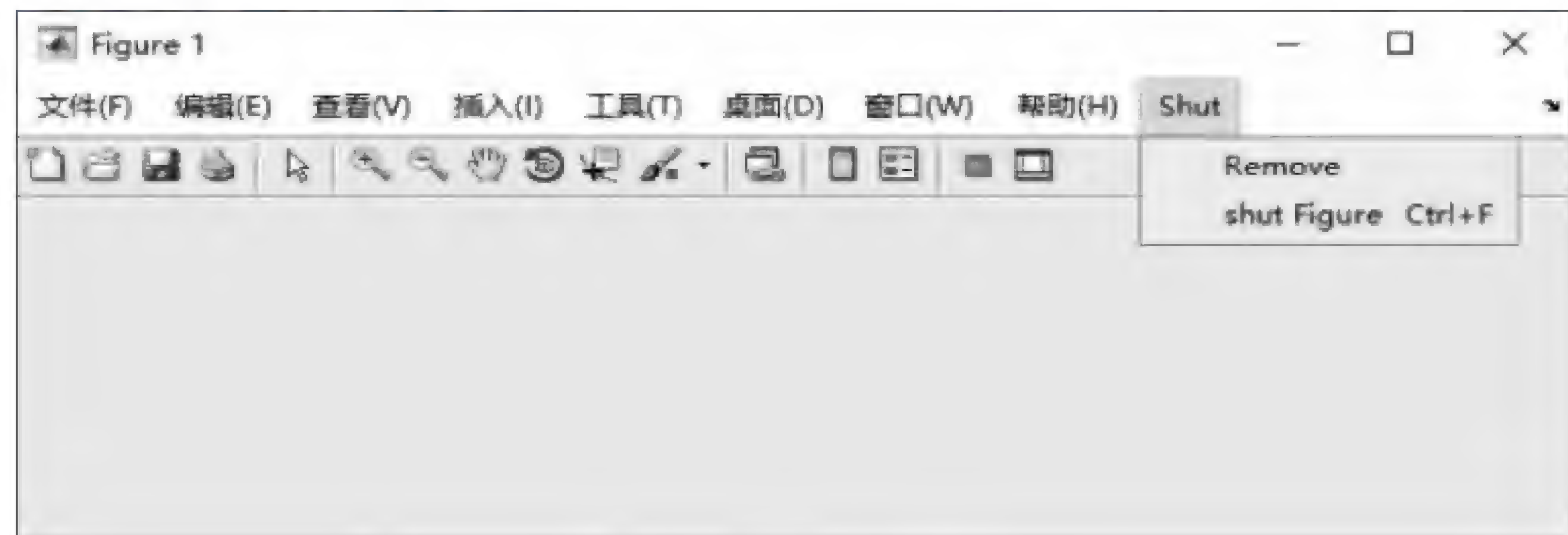


图 4-9 快捷菜单图形

当用户需要激活快捷菜单时，可以在选择如图 4-9 所示窗口后，按 Alt+S 组合键，运行效果如图 4-10 所示。其中，阴影部分表示已经被用户选中。

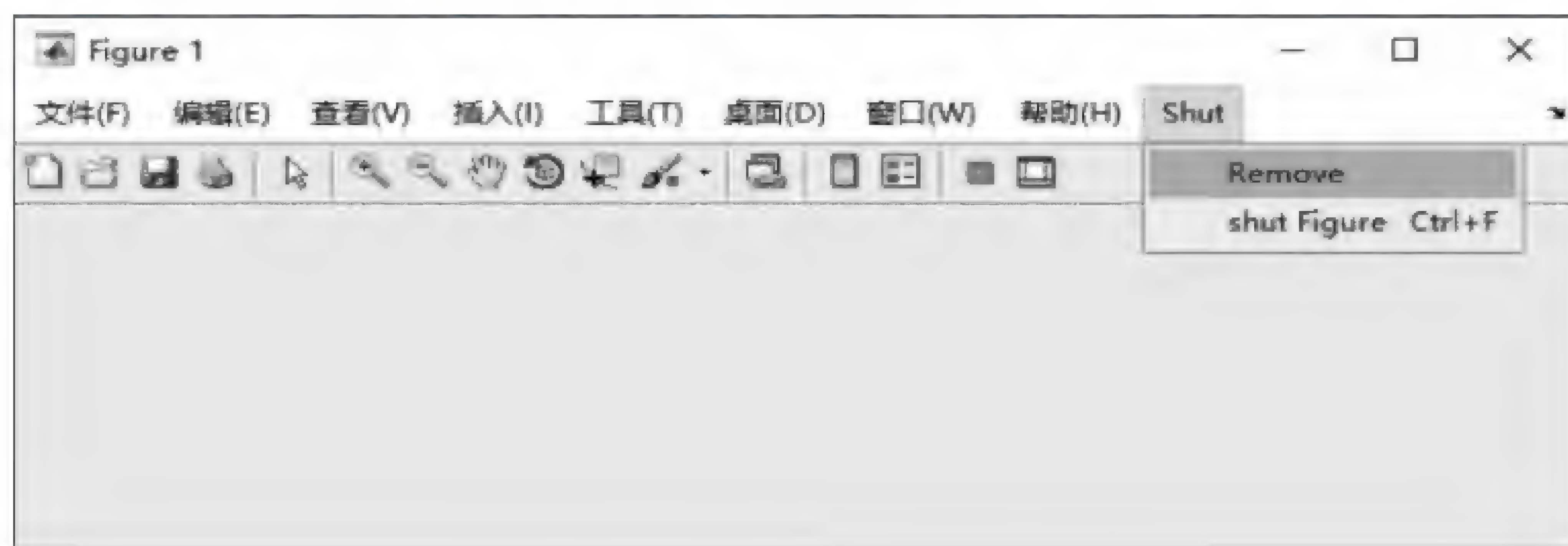


图 4-10　选中 Remove 选项

在图 4-10 中，如果需要产生 Remove 的效果，需要再按组合键 Alt+R；如果需要产生 shut Figure 的效果，需要再按组合键 Ctrl+F。

4.3　GUIDE 的使用

在 MATLAB 中，GUIDE 是一个组件布局工具集，能够生成用户所需要的组件并将其保存在一个 FIG 文件中。

在 MATLAB 的命令行窗口中输入：

```
guide
```

可以得到 GUIDE 启动界面，如图 4-11 所示。



图 4-11　GUIDE 启动界面

在图 4-11 中保存默认选择，单击“确定”按钮，可以得到 GUIDE 编辑框，一般编辑框的默认名称为 untitled.fig，如图 4-12 所示。

注意：如果系统中已经存在 GUIDE 编辑框，那么后续生成的 GUIDE 编辑框默认名

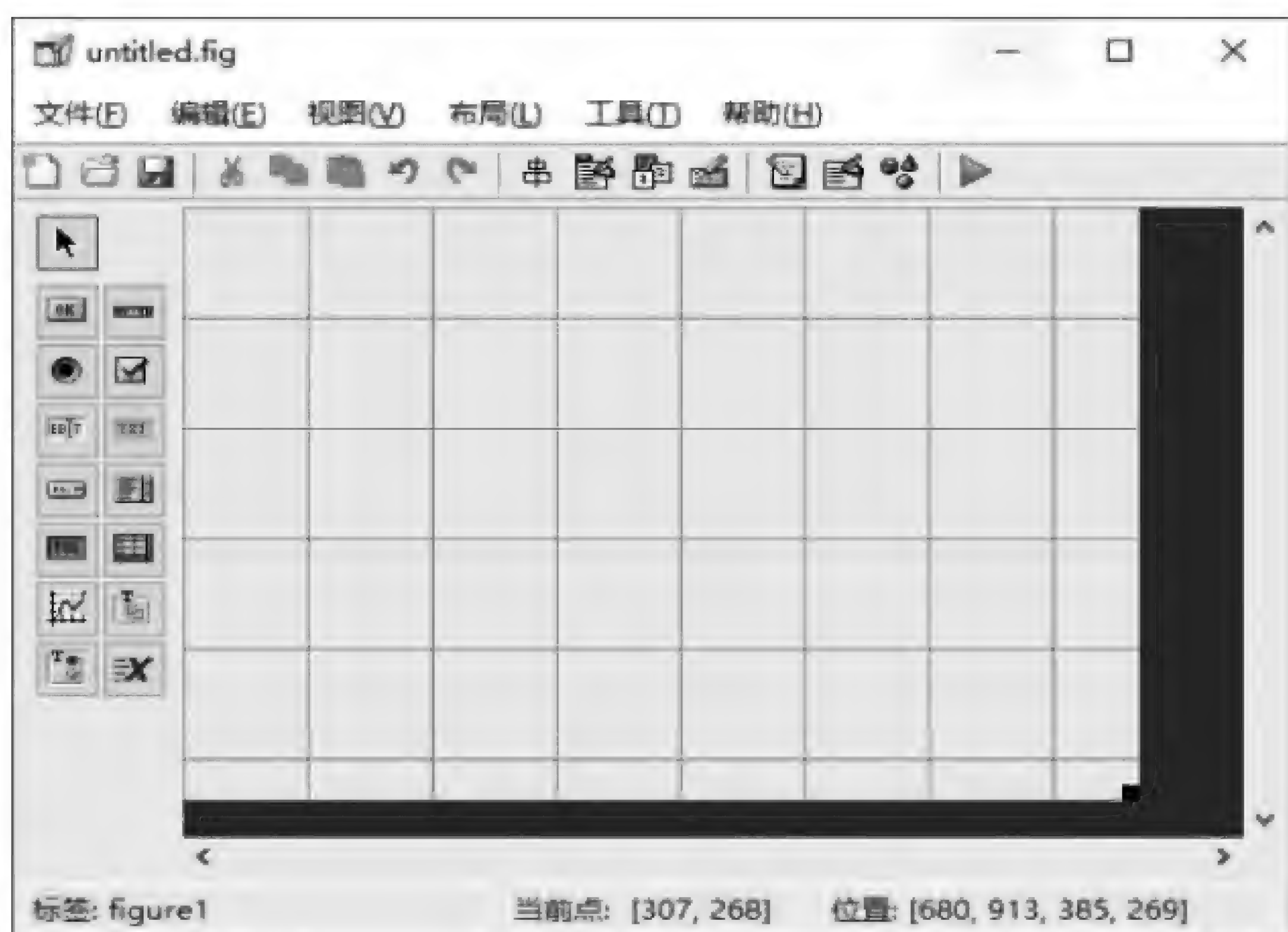


图 4-12　GUIDE 编辑框

称将依次为 untitled1. fig、untitled2. fig、untitled3. fig 等。

GUIDE 是一个界面设计工具，这些工具包括以下四个部分：

（1）界面设计编辑器：添加并排列图形窗口中的组件对象；

（2）属性检查器：检查并设置组件的属性值；

（3）对象浏览器：观察并设置组件的属性值；

（4）菜单编辑器：创建窗口菜单和文本菜单。

以上这些工具通过界面设计编辑器的相应菜单进行调用。

4.4　使用 M 文件创建 GUI 对象

除了使用 GUIDE 创建 GUI 对象外也可以使用 M 文件创建 GUI 对象。

本节将介绍如何使用 M 文件来创建一个简单的 GUI 对象，该 GUI 对象中不包含 GUI 菜单和控件，其与用户之间的互动通过键盘和鼠标操作来实现。对于这种类型的 GUI 对象，最好使用 M 文件来直接编写，而不适用于使用 GUIDE 来创建。

【例 4-3】 对于传递函数为 $G=\frac{1}{s^2+2\zeta s+1}$的二阶系统，制作一个能绘制该系统单位阶跃响应的图形用户界面。

解：首先编写 MATLAB 代码生成图形界面。

```
clear all
clc
H = axes('unit','normalized','position',[0,0,1,1],'visible','off');
set(gcf,'currentaxes',H);
str = '\fontname{隶书}二阶系统的阶跃响应曲线';
text(0.12,0.93,str,'fontsize',13);
```

```
h_fig = get(H,'parent');
set(h_fig,'unit','normalized','position',[0.1,0.2,0.7,0.4]);
h_axes = axes('parent',h_fig,'unit','normalized','position',[0.1,0.15,0.55,0.7],...
'xlim',[0 15],'ylim',[0 1.8],'fontsize',8);
```

运行函数得到如图 4-13 所示界面。



图 4-13　图形界面

然后编写代码生成静态文本和编辑框。

```
h_text = uicontrol(h_fig,'style','text',...
'unit','normalized','position',[0.67,0.73,0.25,0.14],...
'horizontal','left','string',{'输入阻尼系数','zlxs = '});
h_edit = uicontrol(h_fig,'style','edit',...
'unit','normalized','position',[0.67,0.59,0.25,0.14],...
'horizontal','left',...
'callback',[...
'z = str2num(get(gcbo,''string''));',...
't = 0:0.1:15;',...
'for k = 1:length(z);',...
'y(:,k) = step(1,[1 2 * z(k) 1],t);',...
'plot(t,y(:,k));',...
'if (length(z)> 1) ,hold on,end,',...
'end;',...
'hold off,']);
```

运行函数得到如图 4-14 所示的静态文本和编辑框。

形成坐标方格控制键，编写如下程序：



图 4-14　静态文本和编辑框

```
h_push1 = uicontrol(h_fig,'style','push',...
'unit','normalized','position',[0.67,0.37,0.12,0.15],...
'string','grid on','callback','grid on');
h_push2 = uicontrol(h_fig,'style','push',...
'unit','normalized','position',[0.67,0.15,0.12,0.15],...
'string','grid off','callback','grid off');
```

运行函数得到如图 4-15 所示的坐标方格控制键。



图 4-15　坐标方格控制键的形成

输入阻尼系数 0.5 和 0.02：0.01：3，分别得到如图 4-16 和图 4-17 所示图形。

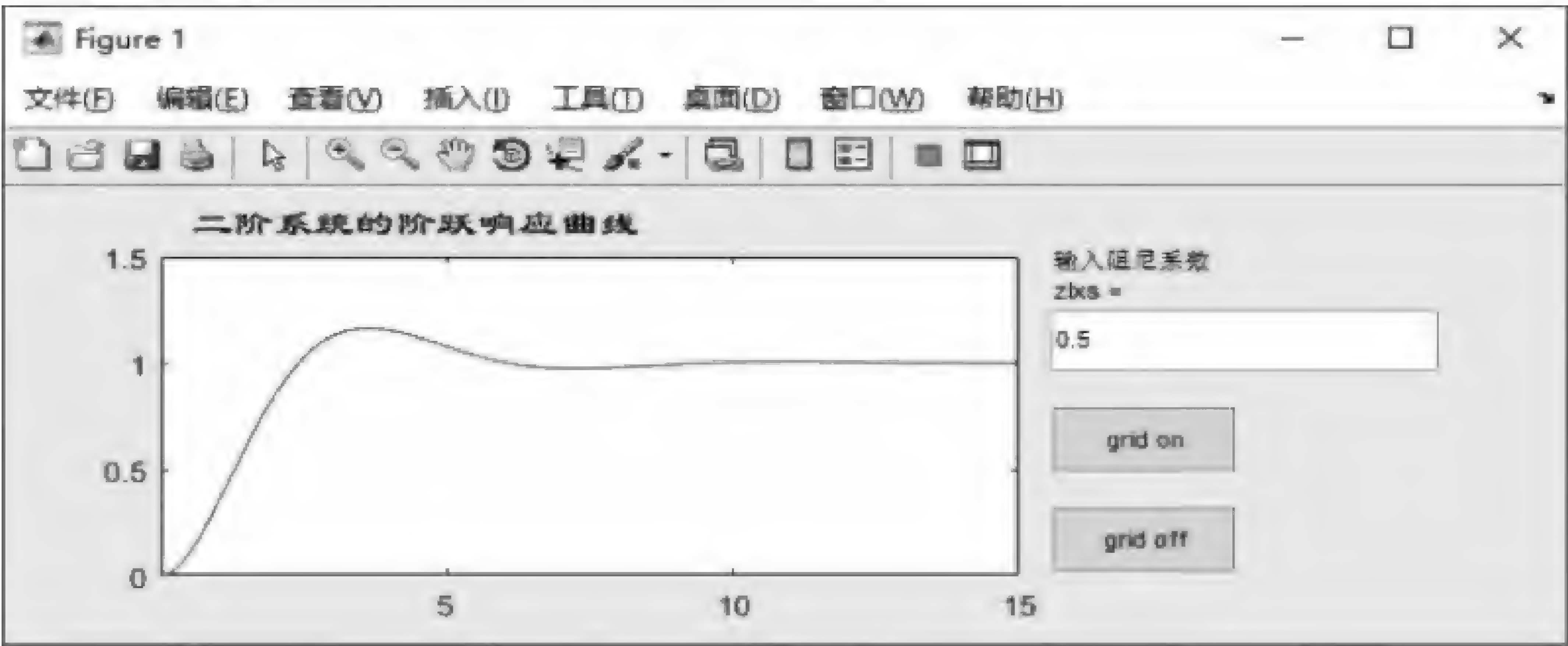


图 4-16 使用该界面

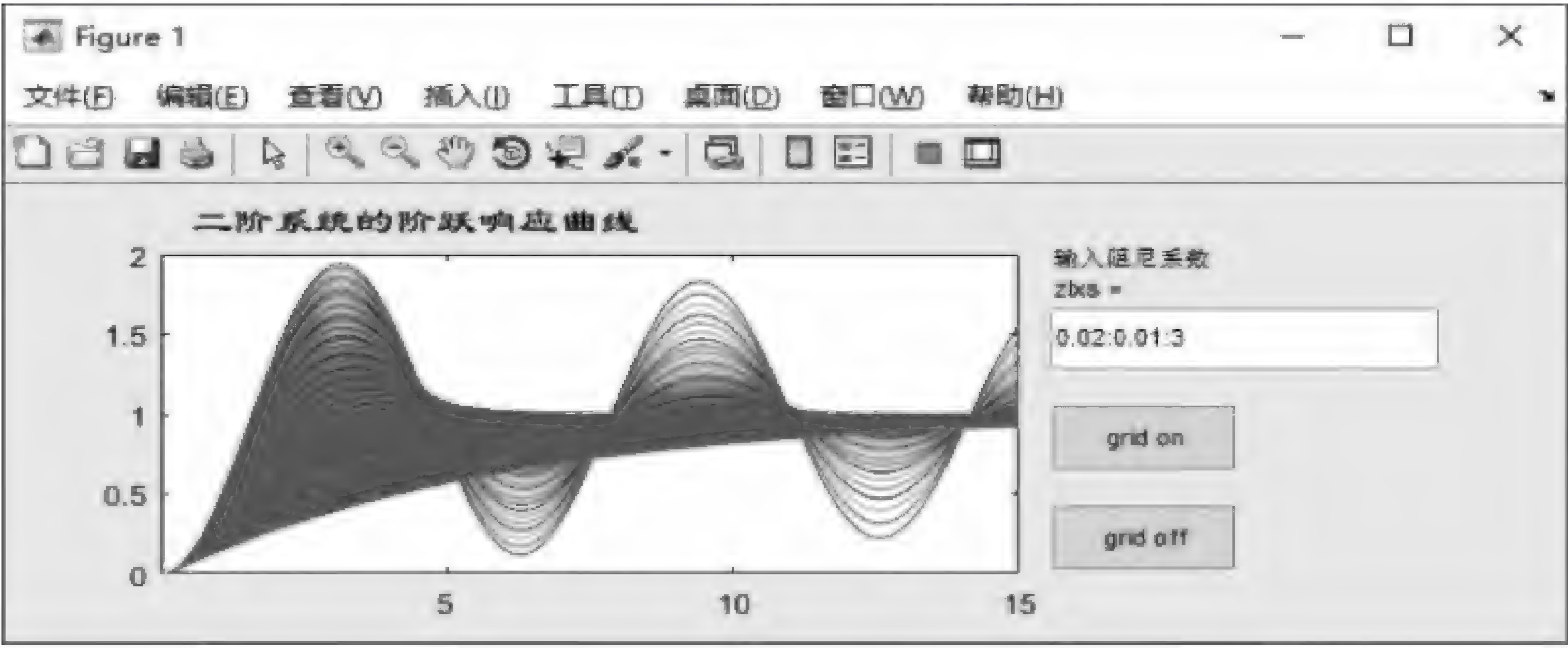
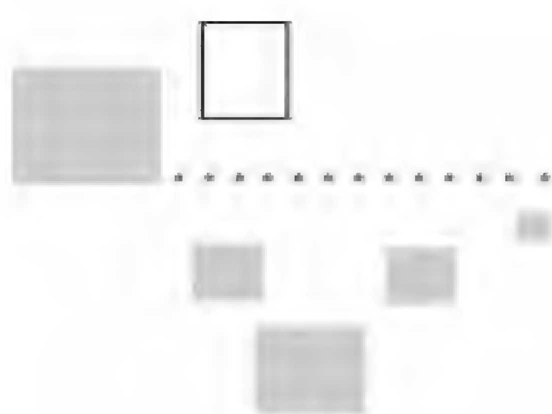


图 4-17 多种相应曲线

本章小结

本章首先对 GUI 设计的基础概念做了简单介绍，然后介绍了创建 GUI 菜单的几种方法，并对菜单属性做了简单介绍。最后举例说明使用 M 文件创建 GUI 对象。



第二部分 MATLAB常规优化算法

第5章 MATLAB线性规划

线性规划是运筹学中研究较早、发展较快、应用广泛且方法较成熟的一个重要分支，它是辅助人们进行科学管理的一种数学方法。

学习目标：

(1) 了解 MATLAB 线性规划基本概念；

(2) 了解 MATLAB 线性规划标准形式；

(3) 掌握 MATLAB 中线性规划函数的应用；

(4) 熟练掌握 MATLAB 线性规划问题求解。

5.1 线性规划的概念

线性规划是研究线性约束条件下线性目标函数极值问题的数学理论和方法，英文缩写为 LP。它是运筹学的一个重要分支，广泛应用于军事作战、经济分析、经营管理和工程技术等方面。为合理地利用有限的人力、物力、财力等资源做出最优决策，提供科学的依据。

线性规划模型首先是列出约束条件及目标函数，然后画出约束条件所表示的可行域，最后在可行域内求目标函数的最优解及最优值。线性规划模型求解流程图如图 5-1 所示。

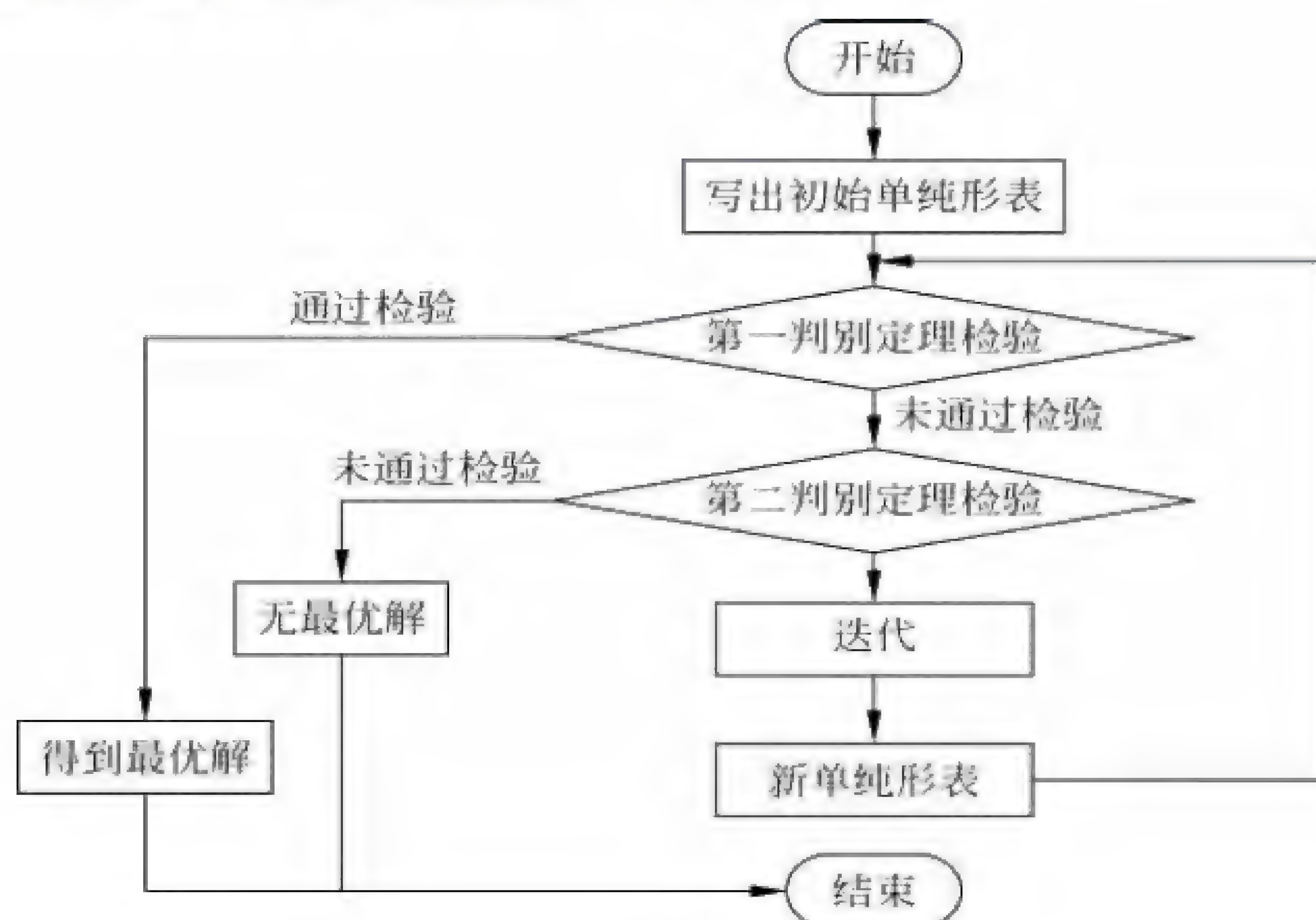


图 5-1 线性规划模型求解流程图

5.2 线性规划的标准形式

线性规划方法是在第二次世界大战中发展起来的一种重要的数学方法，它是处理线性目标函数和线性约束的一种较为成熟的方法，主要用于研究有限资源的最佳分配问题，即如何对有限的资源作出最佳方式地调配和最有利地使用，以便最充分地发挥资源的效能去获取最佳的经济效益。目前已经广泛应用于军事、经济、工业、农业、教育、商业和社会科学等方面。

线性规划问题的标准形式是

$$\begin{cases} \min z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\ x_1, x_2, \cdots, x_n \geqslant 0 \end{cases}$$

或

$$\begin{cases} \min z = \sum_{j=1}^{n} c_jx_j \\ \sum_{j=1}^{n} a_{ij}x_j = b_i, i = 1,2,\cdots,m \\ x_j \geqslant 0, j = 1,2,\cdots,n \end{cases}$$

写成矩阵形式为

$$\begin{cases} \min z = \boldsymbol{CX} \\ \boldsymbol{AX} = \boldsymbol{b} \\ \boldsymbol{X} \geqslant 0 \end{cases}$$

线性规划的标准形式要求使目标函数最小化，约束条件取等式，变量 b 非负。不符合这几个条件的线性模型可以转化为标准形式。

从实际问题中建立数学模型一般有以下三个步骤：

(1) 根据影响所要达到目的的因素找到决策变量；

(2) 由决策变量和所要达到目的之间的函数关系确定目标函数；

(3) 由决策变量所受的限制条件确定决策变量所要满足的约束条件。

所建立的数学模型具有以下特点：

(1) 每个模型都有若干个决策变量($x_1, x_2, x_3, \cdots, x_n$)，其中 n 为决策变量个数。决策变量的一组值表示一种方案，同时决策变量一般是非负的。

(2) 目标函数是决策变量的线性函数，根据具体问题可以是最大化(max)或最小化(min)，二者统称为最优化(opt)。

(3) 约束条件也是决策变量的线性函数。

当得到的数学模型的目标函数为线性函数，约束条件为线性等式或不等式时，称此

数学模型为线性规划模型。

5.3 线性规划的 MATLAB 函数

在 MATLAB 中,用于 LP 的求解函数为 linprog,其调用格式为

```
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
[x,fval] = linprog(...)
[x,fval,exitflag] = linprog(...)
[x,fval,exitflag,output] = linprog(...)
[x,fval,exitflag,output,lambda] = linprog(...)
```

其中,f、A、b 是不可默认的输入变量,x 是不可默认的输出变量,它是问题的解。lb、ub 均是向量,分别表示 x 的下界和上界,x0 为 x 的起始点,options 为 optimset 函数中定义的参数的值,fval 是目标函数在解 x 处的值,lambda 为在解 x 处的 Lagrange 乘子。

lambda 参数的属性如下:

lambda. lower: lambda 的下界。

lambda. upper: lambda 的上界。

lambda. ineqlin: lambda 的线性不等式。

lambda. eqlin: lambda 的线性等式。

函数 linprog 的具体用例解释如下:

x=linprog(f,A,b): 求解问题 $\min f^T x$,约束条件为 $Ax \leqslant b$。

x=linprog(f,A,b,Aeq,beq): 求解上面的问题,但增加等式约束,即 Aeqx=beq。若没有不等式存在,则令 A=[]、b=[]。

x=linprog(f,A,b,Aeq,beq,lb,ub): 定义设计变量 x 的下界 lb 和上界 ub,使得 x 始终在该范围内。若没有等式约束,令 Aeq=[]、beq=[]。

x=linprog(f,A,b,Aeq,beq,lb,ub,x0): 设置初值为 x0。该选项只适用于中型问题,默认大型算法将忽略初值。

x=linprog(f,A,b,Aeq,beq,lb,ub,x0,options): 用 options 指定的优化参数进行最小化。

[x,fval]=linprog(…): 返回解 x 处的目标函数值 fval。

[x,lambda,exitflag]=linprog(…): 返回 exitflag 值,描述函数计算的退出条件。

[x,lambda,exitflag,output]=linprog(…): 返回包含优化信息的输出变量 output。

[x,fval,exitflag,output,lambda]=linprog(…): 将解 x 处的 Lagrange 乘子返回到 lambda 参数中。

【例 5-1】 有以下模型：

$$\min Z = -4a + b + 7c$$

$$\text{s.t.}\begin{cases} a + b - c = 5 \\ 3a - b + c \leqslant 4 \\ a + b - 4c \leqslant -7 \\ a, b \geqslant 0 \end{cases}$$

请问 a,b,c 分别取何值时，Z 有最小值？

解：根据题中模型，编写以下代码：

```
clear all
clc
c = [ - 4 1 7];
A = [3  - 1 1;1 1  - 4];
b = [4;  - 7];
Aeq = [1 1  - 1];
beq = [5];vlb = [0, 0];
vub = [ ];
[x, fval] = linprog(c, A, b, Aeq, beq, vlb, vub)
```

运行后得到结果如下：

```
Optimization terminated.

x =
     2.2500
     6.7500
     4.0000

fval =
   25.7500
```

即 a,b,c 分别取 2.2500、6.7500、4.0000 时，Z 有最小值 25.7500。

【例 5-2】 根据以下模型：

$$\begin{cases} x(1) + x(2) \leqslant 2 \\ x(1) + x(2)/4 \leqslant 1 \\ x(1) - x(2) \leqslant 2 \\ -x(1)/4 - x(2) \leqslant 1 \\ -x(1) - x(2) \leqslant -1 \\ -x(1) + x(2) \leqslant 2 \end{cases}$$

求解 $\boldsymbol{X}$ 值。

解：根据题意，编写代码如下：

```
clear all
clc
A = [  1     1
       1   1/4
       1    -1
    -1/4  -1
     -1    -1
     -1     1];

b = [2  1  2  1  -1  2];

Aeq = [1  1/4];
beq = 1/2;

lb = [-1,-0.5];
ub = [1.5,1.25];
f = [-1  -1/3];

x = linprog(f,A,b,Aeq,beq,lb,ub)
```

运行后得到结果如下：

```
Optimization terminated.
x =
    0.1875
    1.2500
```

【例 5-3】 求函数 $f(x)=-5x_1-4x_2-6x_3$ 的最小值并且满足以下条件：

$$\begin{cases} x_1-x_2+x_3\leqslant 20 \\ 3x_1+2x_2+4x_3\leqslant 42 \\ 3x_1+2x_2\leqslant 30 \\ x_1\geqslant 0 \\ x_2\geqslant 0 \\ x_3\geqslant 0 \end{cases}$$

解：根据目标函数及限制条件，编写代码如下：

```
clear all
clc

f = [-5; -4; -6];

A = [ 1   -1  1
      3    2  4
      3    2  0];
b = [20;42;30];

lb = zeros(3,1);
```

```
Aeq = [];
beq = [];

[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb);

x,lambda.ineqlin,lambda.lower

A * x
```

运行后得到结果如下：

```
Optimization terminated.

x =
      0.0000
     15.0000
      3.0000
ans =
      0.0000
      1.5000
      0.5000
ans =
      1.0000
      0.0000
      0.0000
ans =
    - 12.0000
      42.0000
      30.0000
```

5.4 线性规划问题求解方法

求解线性规划问题的基本方法是单纯形法。为了提高解题速度，又有改进单纯形法、对偶单纯形法、原始对偶方法、分解算法和各种多项式时间算法。对于只有两个变量的简单的线性规划问题，也可采用图解法求解。

线性规划包括单纯形线性规划和多目标线性规划。

5.4.1 单纯形线性规划问题求解

单纯形法是从所有基本可行解的一个较小部分中通过迭代过程选出最优解，其迭代过程的一般描述如下：

(1) 将线性规划化为典范形式，从而可以得到一个初始基本可行解 $x(0)$（初始顶点），将它作为迭代过程的出发点，其目标值为 $z(x(0))$。

(2) 寻找一个基本可行解 $x(1)$，使 $z(x(1))\leqslant z(x(0))$。方法是通过消去法将产生 $x(0)$的典范形式化为产生 $x(1)$的典范形式。

(3) 继续寻找较好的基本可行解 $x(2)$，$x(3)$，…使目标函数值不断改进，即 $z(x(1))\geqslant z(x(2))\geqslant z(x(3))\geqslant\cdots$当某个基本可行解再也不能被其他基本可行解改进时，它就是所求的最优解。

MATLAB 采用投影法求解线性规划问题，该方法是单纯形法的变种。MATLAB 中用来求解线性规划的函数是 linprog。

【例 5-4】 求函数的最小值 $f(x)=5x_1+3x_2-7x_3$，其中 $\boldsymbol{x}$ 满足条件

$$\begin{cases} x_1 - x_2 + x_3 \leqslant 23 \\ 3x_1 + 2x_2 + 4x_3 \leqslant 40 \\ 3x_1 + 2x_2 \leqslant 32 \\ 0 \leqslant x_1, 0 \leqslant x_2, 0 \leqslant x_3 \end{cases}$$

解：首先将变量按顺序排好，然后用系数表示目标函数，即

```
f = [5;3; -7];
```

因为没有等式条件，所以 Aeq、beq 都是空矩阵，即

```
Aeq = [ ];
beq = [ ];
```

不等式条件的系数为

$$\boldsymbol{A} = \begin{bmatrix} 1 & -1 & 1 \\ 3 & 2 & 4 \\ 3 & 2 & 0 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 23 \\ 40 \\ 32 \end{bmatrix}$$

由于没有上限要求，故 lb、ub 设为

$$\boldsymbol{lb} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{ub} = \begin{bmatrix} \text{inf} \\ \text{inf} \\ \text{inf} \end{bmatrix}$$

根据以上分析，编写 MATLAB 代码如下：

```
clear all
clc
f = [5; 3; -7];                              % 目标函数的系数
A = [ 1   -1  1
      3    2  4
      3    2  0];
b = [23; 40; 32];
lb = [0;0;0];                                % 各变量的下限
ub = [inf;inf;inf];                          % 各变量的上限
[x,fval] = linprog(f,A,b,[ ],[ ],lb,[ ]); % 求解运算
x
fval
```

运行程序得到结果如下：

```
Optimization terminated.

x =
     0.0000
     0.0000
    10.0000
fval =
    -70.0000
```

【例 5-5】 求解下列优化问题：

$$f(x) = -6x_1 + 5x_2 - 4x_3$$

$$\begin{cases} x_1 - x_2 + x_3 \leqslant 19 \\ 3x_1 + 2x_2 + 4x_3 \leqslant 36 \\ 3x_1 + 2x_2 \leqslant 25 \\ 0 \leqslant x_1, \quad 0 \leqslant x_2, \quad 0 \leqslant x_3 \end{cases}$$

解：在 MATLAB 命令窗口输入以下代码：

```
clear all
clc
f = [ - 6;5; - 4];
A = [1  - 1 1;3 2 4;3 2 0];
b = [19;36;25];
lb = zeros(3,1);
[x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)
```

运行代码得到结果如下：

```
Optimization terminated.

x =
    8.3333
    0.0000
    2.7500
fval =
   - 61.0000
exitflag =
     1

output =
  包含以下字段的 struct:
         iterations: 5
          algorithm: 'interior - point - legacy'
       cgiterations: 0
            message: 'Optimization terminated.'
    constrviolation: 0
```

```
      firstorderopt: 2.7701e-07

lambda =
  包含以下字段的 struct:
    ineqlin: [3×1 double]
      eqlin: [0×1 double]
      upper: [3×1 double]
      lower: [3×1 double]
```

exitflag = 1 表示过程正常收敛于解 x 处。

5.4.2 多目标线性规划问题求解

多目标线性规划是多目标最优化理论的重要组成部分，由于多个目标之间的矛盾性和不可公度性，要求使所有目标均达到最优解是不可能的，因此多目标规划问题往往只是求其有效解。

目前求解多目标线性规划问题有效解的方法包括理想点法、线性加权和法、最大最小法、目标规划法。

多目标线性规划有两个以及两个以上的目标函数，且目标函数和约束条件全是线性函数，其数学模型表示为

$$\max\begin{cases} z_1 = c_{11}x_1 + c_{12}x_2 + \cdots + c_{1n}x_n \\ z_2 = c_{21}x_1 + c_{22}x_2 + \cdots + c_{2n}x_n \\ \quad\vdots \\ z_r = c_{r1}x_1 + c_{r2}x_2 + \cdots + c_{rn}x_n \end{cases}$$

约束条件为

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leqslant b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leqslant b_2 \\ \quad\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leqslant b_m \\ x_1, x_2, \cdots, x_n \geqslant 0 \end{cases}$$

上述多目标线性规划可用矩阵形式表示为

$$\max \boldsymbol{Z} = \boldsymbol{C}x$$

约束条件为

$$\begin{cases} \boldsymbol{A}x \leqslant \boldsymbol{b} \\ x \geqslant 0 \end{cases}$$

1. 理想点法

$\max \boldsymbol{Z} = \boldsymbol{C}x$

在 $\begin{cases} \boldsymbol{A}x \leqslant \boldsymbol{b} \\ x \geqslant 0 \end{cases}$ 中，先求解 r 个单目标问题：$\min\limits_{x\in D} Z_j(x), j=1,2,\cdots,r$。设其最优值为

Z_j^*，称 $\boldsymbol{Z}^*$ 为值域中的一个理想点。于是在期望的某种度量下，寻求距离 $\boldsymbol{Z}^*$ 最近的 $\boldsymbol{Z}$ 作为近似值。一种最直接的方法就是最短距离理想点法，构造评价函数 $\varphi(Z)=\sqrt{\sum_{i=1}^{r}[Z_i-Z_i^*]^2}$，然后极小化 $\varphi[Z(x)]$，即求解 $\min\limits_{x\in D}\varphi[Z(x)]=\sqrt{\sum_{i=1}^{r}[Z_i(x)-Z_i^*]^2}$，$\mathrm{maz}\boldsymbol{Z}=\boldsymbol{C}x$ 并将它的最优解 x^* 作为 $\begin{cases}\boldsymbol{A}x\leqslant\boldsymbol{b}\\ x\geqslant 0\end{cases}$ 在这种意义下的最优解。

【例 5-6】 利用理想点法求解

$$\max f_1(x)=-3x_1-4x_2$$
$$\max f_2(x)=-5x_1+2x_2$$
$$\text{s. t}\begin{cases}2x_1-3x_2\leqslant 19\\ 3x_1+x_2\leqslant 11\\ x_1,x_2\geqslant 0\end{cases}$$

解： 先分别对单目标求解。

求解 $f_1(x)$ 最优解的 MATLAB 代码如下：

```
clear all
clc
f = [ - 3; - 4];
A = [2, - 3;3,1];
b = [19;11];
lb = [0;0];
[x,fval] = linprog(f,A,b,[],[],lb)
```

结果输出如下：

```
Optimization terminated.
x =
     0.0000
    11.0000
fval =
   - 44.0000
```

即最优解为 44。

求解 $f_2(x)$ 最优解的 MATLAB 代码如下：

```
f = [ - 5; - 5];
A = [2, - 3;3,1];
b = [19;11];
lb = [0;0];
[x,fval] = linprog(f,A,b,[],[],lb)
```

结果输出如下：

```
Optimization terminated.
x2 =
     0.0000
    11.0000
fval =
   -22.0000
```

即最优解为22。

于是得到理想点：(44,22)。

然后求如下模型的最优解：

$$\min_{x \in D} \varphi[f(x)] = \sqrt{[f_1(x) - 44]^2 + [f_2(x) - 22]^2}$$

$$\text{s. t} \begin{cases} 2x_1 - 3x_2 \leqslant 19 \\ 3x_1 + x_2 \leqslant 11 \\ x_1, x_2 \geqslant 0 \end{cases}$$

MATLAB代码如下：

```
A = [2, -3;3,1];
b = [19;11];
x0 = [1;1];
lb = [0;0];
x = fmincon('((3 * x(1) - 4 * x(2) - 44)^2 + (5 * x(1) + 2 * x(2) - 22)^2)^(1/2)',x0,A,b,[],[],
lb,[])
```

结果输出如下：

```
x =
     3.6667
     0.0000
```

2. 线性加权和法

在具有多个指标的问题中，人们总希望对那些相对重要的指标给予较大的权系数，因而将多目标向量问题转化为所有目标的加权求和的标量问题。

基于上述设计，构造如下评价函数，即

$$\min_{x \in D} Z(x) = \sum_{i=1}^{r} \omega_i Z_i(x)$$

$$\max \boldsymbol{Z} = \boldsymbol{Cx}$$

将它的最优解 $\boldsymbol{x}^*$ 作为 $\begin{cases} \boldsymbol{A}x \leqslant \boldsymbol{b} \\ x \geqslant 0 \end{cases}$ 在线性加权和意义下的最优解。其中 ω_i 为加权因子，其选取的方法很多，有专家打分法、容限法和加权因子分解法等。

【例 5-7】 对例5-6采用线性加权和法求解。(权系数分别取 $\omega_1 = 0.5, \omega_2 = 0.5$)

解：构造如下评价函数，即求如下模型的最优解：

$$\min\{-4\times(-3x_1+4x_2)+1\times(-5x_1-2x_2)\}$$
$$\text{s.t}\begin{cases}2x_1-3x_2\leqslant 19\\3x_1+x_2\leqslant 11\\x_1,x_2\geqslant 0\end{cases}$$

MATLAB 代码如下：

```
clear all
clc
f = [ - 4;1];
A = [2, - 3;3,1];
b = [19;11];
lb = [0;0];
x = linprog(f,A,b,[],[],lb)
```

结果输出如下：

```
Optimization terminated.

x =
    3.6667
    0.0000
```

3. 最大最小法

在决策的时候，采取保守策略是稳妥的，即在最坏的情况下，寻求最好的结果，按照此想法，可以构造如下评价函数，即

$$\varphi(Z)=\max_{1\leqslant i\leqslant r} Z_i$$

然后求解

$$\min_{x\in D}\varphi[Z(x)]=\min_{x\in D}\max_{1\leqslant i\leqslant r} Z_i(x)$$
$$\max \boldsymbol{Z}=\boldsymbol{Cx}$$

并将它的最优解 $\boldsymbol{x}^*$ 作为 $\begin{cases}\boldsymbol{Ax}\leqslant\boldsymbol{b}\\\boldsymbol{x}\geqslant 0\end{cases}$ 在最大最小意义下的最优解。

【例 5-8】 对例 5-6 进行最大最小法求解。

解：首先编写目标函数的 M 文件：

```
function f = ex58(x)
f(1) = - 3 * x(1) + 4 * x(2);
f(2) = - 5 * x(1) - 2 * x(2);
```

然后编写 MATLAB 代码如下：

```
clear all
clc
x0 = [1;1];
```

```
A = [2, - 3;3,1];
b = [19;11];
lb = zeros(2,1);
[x,fval] = fminimax('ex58',x0,A,b,[],[],lb,[])
```

结果输出如下：

```
x =
    3.6667
    0.0000
fval =
  - 11.0000   - 18.3333
```

多目标线性规划是优化问题的一种，由于其存在多个目标，要求各目标同时取得较优的值，使得求解的方法与过程都相对复杂。通过将目标函数进行模糊化处理，可将多目标问题转化为单目标，借助工具软件，从而达到较易求解的目标。

5.5 线性规划实例

在企业的各项管理活动中，如计划、生产、运输、技术等问题，线性规划是指从各种限制条件的组合中，选出最为合理的计算方法，建立线性规划模型从而求得最佳结果。本节将从企业活动中的实例出发讲解线性规划的求解。

5.5.1 生产决策问题

【例 5-9】 某厂生产甲、乙两种产品，已知制成一吨产品甲需要资源 A 5t，资源 B 4m^3，资源 C 1 个单位；制成 1t 产品乙需要资源 A 2t，资源 B 6m^3，资源 C 7 个单位。若 1t 产品甲和乙的经济价值分别为 9 万元和 4 万元，3 种资源的限制量分别为 85t、210m^3 和 250 个单位。试分析应生产这两种产品各多少吨才能使创造的总经济价值最高？

解：这里可以令生产产品甲的数量为 x_1，生产产品乙的数量为 x_2。根据题意，编码代码如下：

```
clear all
clc
f = [ - 9; - 4];
A =[5 2
      4 7
      1 6];
b = [85; 210; 250];
lb = zeros(2,1);
```

然后调用 linprog 函数：

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)
```

最优化结果如下：

```
Optimization terminated.

x =
    6.4815
   26.2963
fval =
 -163.5185
exitflag =
     1
output =
  包含以下字段的 struct:
         iterations: 5
          algorithm: 'interior-point-legacy'
       cgiterations: 0
            message: 'Optimization terminated.'
    constrviolation: 0
      firstorderopt: 1.6144e-11

lambda =
  包含以下字段的 struct:
    ineqlin: [3×1 double]
      eqlin: [0×1 double]
      upper: [2×1 double]
      lower: [2×1 double]
```

由上可知，生产甲种产品 6.4815t、乙种产品 26.2963t 可使创造的总经济价值最高，最高经济价值为 163.5185 万元。exitflag=1 表示过程正常收敛于解 x 处。

5.5.2 工作人员计划安排问题

【例 5-10】 某昼夜服务的公共交通系统每天各时间段（每 4 小时为一个时间段）所需的值班人数如表 5-1 所示，这些值班人员在某一时段开始上班后要连续工作 8 小时（包括轮流用餐时间），问该公共交通系统至少需要多少名工作人员才能满足值班的需要？

表 5-1 各时段所需值班人数

班次	时间段	所需人数
1	5:00～9:00	40
2	9:00～13:00	35
3	13:00～17:00	60
4	17:00～21:00	70
5	21:00～1:00	45
6	1:00～5:00	30

解：这里可以设 x_i 为第 i 个时段开始上班的人员数。

根据题意，编写代码如下：

```
clear all
clc
f = [1;1;1;1;1;1];
A = [ -1 0 0 0 0 -1
    -1 -1 0 0 0 0
   0 -1 -1 0 0 0
   0 0 -1 -1 0 0
   0 0 0 -1 -1 0
   0 0 0 0 -1 -1];
b = [ -40; -35; -60; -70; -45; -30];
lb = zeros(6,1);
```

然后调用 linprog 函数

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)
```

最优化结果如下：

```
Optimization terminated.

x =
   20.6959
   18.1954
   41.8046
   30.6180
   14.3820
   19.3041
fval =
  145.0000
exitflag =
     1
output =
  包含以下字段的 struct:
         iterations: 5
          algorithm: 'interior-point-legacy'
       cgiterations: 0
            message: 'Optimization terminated.'
    constrviolation: 0
      firstorderopt: 1.8243e-07

lambda =
  包含以下字段的 struct:
    ineqlin: [6×1 double]
      eqlin: [0×1 double]
```

```
        upper: [6×1 double]
        lower: [6×1 double]
```

可见，只要6个时段分别安排21人、18人、42人、31人、14人和19人就可以满足值班的需要，共计145人，并且计算结果 exitflag =1 是收敛的。

5.5.3 投资问题

【例 5-11】 某单位有一批资金用于4个工程项目的投资，用于各工程项目时所得的净收益(投入资金的百分比)如表5-2所示。

表 5-2 工程项目收益

工程项目	A	B	C	D
收益(%)	18	10	9	12

由于某种原因，决定用于项目A的投资不大于其他各项投资之和；而用于项目B和C的投资要大于项目D的投资。试确定使该单位收益最大的投资分配方案。

解：这里可以用 x_1、x_2、x_3 和 x_4 分别代表用于项目A、B、C和D的投资百分数，由于各项目的投资百分数之和必须等于100%，所以 $x_1+x_2+x_3+x_4=1$

根据题意，编写代码如下：

```
clera all
clc
f = [-0.18;-0.1;-0.09;-0.12];
A = [1 -1 -1 -1
      0 -1 -1 1];
b = [0; 0];
Aeq=[1 1 1 1];
beq=[1];
lb = zeros(4,1);
```

然后调用 linprog 函数

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb)
```

结果如下：

```
Optimization terminated.

x =
    0.5000
    0.2500
    0.0000
    0.2500
fval =
```

```
    -0.1450
exitflag =
     1
output =
  包含以下字段的 struct:
         iterations: 9
          algorithm: 'interior-point-legacy'
       cgiterations: 0
            message: 'Optimization terminated.'
    constrviolation: 4.4409e-16
      firstorderopt: 2.4035e-10

lambda =
  包含以下字段的 struct:
    ineqlin: [2×1 double]
      eqlin: 0.1450
      upper: [4×1 double]
      lower: [4×1 double]
```

上面的结果说明，项目 A、B、C、D 投入资金的百分比分别为 50%、25%、0、25%时，该单位收益最大。

5.5.4 工件加工任务分配问题

【例 5-12】 某车间有两台机床甲和乙，可用于加工 3 种工件。假定这两台机床的可用台时数分别为 500 和 800，3 种工件的数量分别为 300、600 和 700，且已知用两台不同机床加工单位数量的不同工件所需的台时数和加工费用如表 5-3 所示。问怎样分配机床的加工任务，才能既满足加工工件的要求，又使总加工费用最低？

表 5-3 机床加工情况

机床类型	单位工作所需加工台时数/小时			单位工件的加工费用/元			可用台时数/小时
	工件 1	工件 2	工件 3	工件 1	工件 2	工件 3	
甲	0.4	1.1	1.4	13	11	12	600
乙	0.5	1.2	1.3	13	10	9	900

解：这里可以设在甲机床上加工工件 1、2 和 3 的数量分别为 x_1、x_2 和 x_3，在乙机床上加工工件 1、2 和 3 的数量分别为 x_4、x_5 和 x_6。根据 3 种工种的数量限制，则有

$$x_1 + x_4 = 400 \quad (\text{对工件 } 1)$$

$$x_2 + x_5 = 600 \quad (\text{对工件 } 2)$$

$$x_3 + x_6 = 500 \quad (\text{对工件 } 3)$$

根据题意，编写代码如下：

```
clear all
clc
```

```
f = [13;11;12;13;10;9];
A = [0.4 1.1 1.4 0 0 0
       0 0 0 0.5 1.2 1.3];
b = [500; 800];
Aeq = [1 0 0 1 0 0
      0 1 0 0 1 0
      0 0 1 0 0 1];
beq = [300 600 700];
lb = zeros(6,1);
```

然后调用 linprog 函数

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb)
```

结果如下：

```
x =
  349.3976
  599.9658
    0.0080
    0.0003
    0.0340
  699.9920

fval =
   1.7442e+04
exitflag =
    -2

output =
  包含以下字段的 struct:
         iterations: 8
          algorithm: 'interior-point-legacy'
       cgiterations: 0
            message: 'Exiting: One or more of the residuals, duality gap, or total relative
error…'
    constrviolation: 299.7327
      firstorderopt: 2.7307e+21

lambda =
  包含以下字段的 struct:
    ineqlin: [2×1 double]
      eqlin: [3×1 double]
      upper: [6×1 double]
      lower: [6×1 double]
```

可见，在甲机床上加工 349 个工件 1、600 个工件 2，在乙机床上加工 700 个工件 3 时，可在满足条件的情况下使总加工费用最小。最小费用为 17442 元，收敛正常。

5.5.5 厂址选择问题

【例 5-13】 A、B、C 三地,每地都出产一定数量的产品,也消耗一定数量的原料,如表 5-4 所示。已知制成每吨产品需 3t 原料,各地之间的距离如下:A—B,150km;A—C,100km;B—C,200km。假定每万吨原料运输 1km 的运价是 5000 元,每万吨产品运输 1km 的运价是 6000 元。由示于地区条件的差异,在不同地点设厂的生产费用也不同。试问究竟在哪些地方设厂,规模多大,才能使总费用最小?另外,由于其他条件限制,在 B 处建厂的规模(生产的产品数量)不能超过 7 万 t。

表 5-4 A、B、C 三地出产产品、消耗原料情况

地点	年产原料/万 t	年销产品/万 t	生产费用/(万元/万 t)
A	22	7	150
B	16	14	120
C	25	0	100

解:这里可令 x_{ij} 为由 i 地运到 j 地的原料数量(万 t),y_{ij} 为由 i 地运往 j 地的产品数量(万 t),$i,j=1,2,3$(分别对应 A、B、C 三地)。

根据题意,编写代码如下:

```
clear all
clc
f = [75;75;50;50;100;100;150;240;210;120;160;220];
A=[1 -1 1 -1 0 0 3 3 0 0 0 0
  -1 1 0 0 1 -1 0 0 3 3 0 0
   0 0 -1 1 -1 1 0 0 0 0 3 3
   0 0 0 0 0 0 0 0 1 1 0 0];
b=[22;16;25;7];
Aeq=[0 0 0 0 0 0 1 0 1 0 1 0
     0 0 0 0 0 0 0 1 0 1 0 1];
beq=[7;14];
lb = zeros(12,1);
```

然后调用 linprog 函数

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,lb)
```

结果如下:

```
Optimization terminated.

x =
     0.0000
     0.0000
     0.0000
     0.0000
```

```
       0.0000
       0.0000
       7.0000
       0.3333
       0.0000
       5.3333
       0.0000
       8.3333
fval =
     3.6033e+03
exitflag =
       1

output =
  包含以下字段的 struct:
         iterations: 8
          algorithm: 'interior-point-legacy'
       cgiterations: 0
            message: 'Optimization terminated.'
    constrviolation: 7.1054e-15
      firstorderopt: 1.4588e-11

lambda =

  包含以下字段的 struct:
    ineqlin: [4×1 double]
      eqlin: [2×1 double]
      upper: [12×1 double]
      lower: [12×1 double]
```

可见要使总费用最小，A、B、C三地的建厂规模分别为7万t、5.3万t和8.3万t。最小总费用为3603.3万元。

5.5.6 确定职工编制问题

【例5-14】 某工厂每日8小时的产量不低于1800件。为了进行质量控制，计划聘请两个不同水平的检验员。一级检验员的速度为25件/h，正确率98%，计时工资5元/h；二级检验员的速度为15件/h，正确率95%，计时工资3元/h。检验员每错检一次，工厂要损失2元。现有可供厂方聘请的检验员人数为一级6人和二级7人。为使总检验费用最省，该工厂应聘请一级、二级检验员各多少名?

解：可以设需要一级和二级检验员的人数分别为x_1名和x_2名。由题意，编写代码如下：

```
clear all
clc
f = [40;36];
```

```
A=[1 0
   0 1
   -5 -3];
b=[6;7;-45];
lb = zeros(2,1);
```

然后调用 linprog 函数

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)
```

结果如下：

```
Optimization terminated.

x =
    6.0000
    5.0000
fval =
  420.0000
exitflag =
     1

output =
  包含以下字段的 struct:
         iterations: 5
          algorithm: 'interior-point-legacy'
       cgiterations: 0
            message: 'Optimization terminated.'
    constrviolation: 0
      firstorderopt: 3.1060e-07

lambda =
  包含以下字段的 struct:

    ineqlin: [3×1 double]
      eqlin: [0×1 double]
      upper: [2×1 double]
      lower: [2×1 double]
```

可见，聘请一级检验员 6 名、二级检验员 5 名可使总检验费用最省，约为 420.00 元，计算收敛。

5.5.7 生产计划的最优化问题

【例 5-15】 某工厂生产 A 和 B 两种产品，它们需要经过 3 种设备的加工，其工时如表 5-5 所示。设备一、二和三每天可使用的时间分别不超过 11 小时、9 小时和 12 小时。产品 A 和 B 的利润随市场的需求有所波动，如果预测未来某个时期内 A 和 B 的利润分

别为 6000 元/t 和 5000 元/t，试问在哪个时期内，每天应生产产品 A、B 各多少 t，才能使工厂获利最大？

表 5-5 生产产品工时

产品	设备一	设备二	设备三
A/(小时/t)	7	5	6
B/(小时/t)	3	3	2
设备每天最多可工作时数/h	12	10	8

解：这里可以设每天应安排生产产品 A 和 B 分别为 x_1t 和 x_2t。

由题意，编写代码如下：

```
clear all
clc
f = [-6;-5];
A=[7 3
   5 3
   6 2];
b=[12;10;8];
lb = zeros(2,1);
```

然后调用 linprog 函数

```
[x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)
```

结果如下：

```
Optimization terminated.

x =
     0.0000
     3.3333
fval =
   -16.6667
exitflag =
      1
output =
  包含以下字段的 struct:
          iterations: 5
           algorithm: 'interior-point-legacy'
        cgiterations: 0
             message: 'Optimization terminated.'
      constrviolation: 0
       firstorderopt: 7.4706e-09
lambda =
```

```
包含以下字段的 struct:
  ineqlin: [3×1 double]
    eqlin: [0×1 double]
    upper: [2×1 double]
    lower: [2×1 double]
```

所以每天生产 A 产品 0t、B 产品 3.3333t 可使工厂获得最大利润 16666.7 元/t。

本章小结

线性规划是运筹学中研究较早、发展较快、应用广泛、方法较成熟的一个重要分支，它是辅助人们进行科学管理的一种数学方法。在经济管理、交通运输、工农业生产等经济活动中，提高经济效果是人们不可缺少的需求，而线性规划所研究的就是在一定条件下，合理安排人力物力等资源，使经济效果达到最好。

本章首先介绍了线性规划的概念和标准形式，然后对 MATLAB 线性规划函数进行了举例说明，最后对线性规划的求解方法和线性规划实例做了详细分析。

第6章 MATLAB非线性规划

非线性规划(nonlinear programming,NP)是具有非线性约束条件或目标函数的数学规划,是运筹学的一个重要分支。非线性规划研究一个 n 元实函数在一组等式或不等式的约束条件下的极值问题,且目标函数和约束条件至少有一个是未知量的非线性函数。

本章介绍了非线性规划和无约束非线性规划基础,并对其MATLAB运用做了举例说明。

学习目标:

(1) 了解非线性规划基础;

(2) 掌握无约束非线性规划的原理及其求解;

(3) 熟练运用非线性规划实例。

6.1 非线性规划基础

非线性规划研究的对象是非线性函数的数值最优化问题,是20世纪50年代形成的一门学科,其理论和应用发展十分迅猛。随着计算机的发展,非线性规划应用越来越广泛,针对不同的问题提出了特别的算法,到目前为止还没有适用于各种非线性规划问题的一般算法,这有待人们进一步研究。

6.1.1 非线性规划标准形式

在实际工作中,常常会遇到目标函数和约束条件中至少有一个是非线性函数的规划问题,即非线性规划问题。由于非线性规划问题在计算上常常是困难的,理论上的讨论也不能像线性规划那样给出简洁的结果形式和全面透彻的结论,这就限制了非线性规划的应用。在数学建模时,要进行认真的分析,对实际问题进行合理的假设、简化。首先考虑用线性规划模型,当线性近似误差较大时,则考虑用非线性规划。

非线性规划问题的标准形式为

$$\min f(x)$$
$$\text{s.t.}\begin{cases} g_i(x) \leqslant 0, i = 1,2,\cdots,m \\ h_j(x) = 0, j = 1,2,\cdots,r \end{cases}$$

其中，$\boldsymbol{x}$ 为 n 维欧式空间 $\mathbf{R}^n$ 中的向量，$f(x)$ 为目标函数，$g_i(x)$、$h_j(x)$ 为约束条件。且 $g_i(x)$、$h_j(x)$、$f(x)$ 中至少有一个是非线性函数。

非线性规划模型按约束条件可分为以下三类：

1. 无约束非线性规划模型

$$\min f(x)$$
$$x \in \mathbf{R}^n$$

2. 等式约束非线性规划模型

$$\min f(x)$$
$$\text{s.t.}\ h_j(x) = 0, \quad j = 1,2,\cdots,r$$

3. 不等式约束非线性规划模型

$$\min f(x)$$
$$\text{s.t.} \quad g_i(x) \leqslant 0, \quad i = 1,2,\cdots,m$$

6.1.2 非线性规划 MATLAB 函数

MATLAB 中用于求解非线性规划的函数为 fmincon，其调用格式如下：

```
x = fmincon(f,x0,A,b)
x = fmincon(f,x0,A,b,Aeq,beq)
x = fmincon(f,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(f,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(f,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
[x,fval] = fmincon(…)
[x, fval, exitflag] = fmincon(…)
[x, fval, exitflag, output] = fmincon(…)
[x, fval, exitflag, output, lambda] = fmincon(…)
```

其中，x0 为初始点，A、b 分别为不等式约束的系数矩阵和右端列向量，lb、ub 分别为变量 x 的下界和上界，options 为指定优化参数进行最小化。

【例 6-1】 求下列非线性规划问题

$$\min\ f(x) = x_1^2 + 2x_2^2 + 7$$
$$\text{s.t.}\begin{cases} x_1^2 - x_2 \geqslant 0 \\ -x_1 - x_2^2 + 3 = 0 \\ x_1, x_2 \geqslant 0 \end{cases}$$

解：根据题意编写 MATLAB 代码如下：

```
clear all
clc
options = optimset;
[x,y] = fmincon('fun1',rand(2,1),[],[],[],[],zeros(2,1),[], ...
'fun2', options)

function f = fun1(x)
f = x(1)^2 + 2 * x(2)^2 + 7;

function [g,h] = fun2(x)
g = - x(1)^2 + x(2);
h = - x(1) - x(2)^2 + 3;
```

运行程序得到结果如下：

```
x =
    1.1640
    1.3550
y =
    12.0269
```

即当 $x_1=1.1640$，$x_2=1.3550$ 时，最小值 $y=12.0269$。

【例 6-2】 求下列非线性规划问题：

$$\min f(x) = 200(x_2 - x_1^2)^2 + (2 - x_1)^2$$

$$\text{s.t.} \begin{cases} x_1 \leqslant 3 \\ x_2 \leqslant 3 \end{cases}$$

解：编写 MATLAB 代码如下：

```
clear all
clc
x0 = [1.1,1.1];
A = [1 0;0 1];
b = [3;3];
[x,fval] = fmincon(@fun1,x0,A,b)

function f = fun1(x)
f = 200 * (x(2) - x(2)^2)^2 + (2 - x(1))^2;
```

运行后得到结果如下：

```
x =

    2.0000   1.0000


fval =

   5.8990e - 14
```

【例 6-3】 求下列非线性规划问题：

$$\min f(x) = e^{x_1}(6x_1^2 + 5x_2^2 + 2x_1x_2 + 4x_2 + 3)$$

$$\text{s.t.} \begin{cases} x_1x_2 - x_1 - x_2 + 3 \leqslant 0 \\ -4x_1x_2 - 3 \leqslant 0 \end{cases}$$

解：编写 MATLAB 代码如下：

```
clear all
clc
x0 = [1,1];
nonlcon = @fun2
[x, fval] = fmincon(@fun1,x0,[],[],[],[],[],[], nonlcon)
```

运行后得到结果如下：

```
x =
    -0.2947   2.5447
fval =
   33.1990
```

6.2 无约束非线性规划

无约束最优化问题在实际应用中也比较常见，如工程中常见的参数反演问题。另外，许多有约束最优化问题可以转换为无约束最优化问题进行求解。

下面介绍有关无约束非线性规划问题的基本数学原理。

6.2.1 基本数学原理

求解无约束最优化问题的方法主要有两类，直接搜索法（search method）和梯度法（gradient method）。

直接搜索法适用于目标函数高度非线性、没有导数或导数很难计算的情况。由于实际工程中很多问题都是非线性的，直接搜索法不失为一种有效的解决办法。常用的直接搜索法为单纯形法，此外还有 Hooke-Jeeves 搜索法、Pavell 共轭方向法等。

在函数的导数可求的情况下，梯度法是一种更优的方法。该法利用函数的梯度（一阶导数）和 Hessian 矩阵（二阶导数）构造算法，可以获得更快的收敛速度。函数 $f(x)$ 的负梯度方向 $-\nabla f(x)$ 反映了函数的最大下降方向，当搜索方向取为负梯度方向时称为最速下降法。常见的梯度法有最速下降法、Newton 法、Marquart 法、共轭梯度法和拟牛顿法（Quasi-Newton method）等。

在所有这些方法中，用得最多的是拟牛顿法，这个方法在每次迭代过程中建立曲率信息，构成二次模型问题。

下面介绍有关 MATLAB 优化工具箱中求解无约束最优化问题的算法。

（1）大型优化算法：若用户在函数中提供梯度信息，则函数将默认选择大型优化算

法，该算法是基于内部映射牛顿法的子空间置信域法。计算中的每一次迭代涉及用 PCG 法求解大型线性系统得到的近似解。

(2) 中型优化算法：fminunc 函数的参数 options. LargeScale 设置为 off。该算法采用的是基于二次和三次混合插值一维搜索法的 BFGS 拟牛顿法。但一般不建议使用最速下降法。

(3) 默认时的一维搜索算法：当 options. LineSearchType 设置为 quadcubic 时，将采用二次和三次混合插值法。将 options. LineSearchType 设置为 cubicpoly 时，将采用三次插值法。第二种方法需要的目标函数计算次数更少，但梯度的计算次数更多。这样，如果提供了梯度信息，或者能较容易地计算出，则三次插值法是更好的选择。

上述涉及的算法局限性主要表现在以下 4 个方面：

(1) 目标函数必须是连续的。fminunc 函数有时会给出局部最优解。

(2) fminunc 函数只对实数进行优化，即 x 必须为实数，而且 $f(x)$ 必须返回实数。当 x 为复数时，必须将它分解为实部和虚部。

(3) 在使用大型算法时，用户必须在 fun 函数中提供梯度(options 参数中 GradObj 属性必须设置为 on)。

(4) 目前若在 fun 函数中提供了解析梯度，则 options 参数 DerivativeCheck 不能用于大型算法以比较解析梯度和有限差分梯度。通过将 options 参数的 MaxIter 属性设置为 0 用中型方法核对导数，然后重新用大型方法求解问题。

6.2.2 无约束非线性规划函数

1. fminunc 函数

该函数用于求多变量无约束函数的最小值，其调用格式如下：

fminunc 给定初值，求多变量标量函数的最小值，常用于无约束非线性最优化问题。

x = fminunc(fun,x0)：给定初值 x0，求 fun 函数的局部极小点 x。x0 可以是标量、向量或矩阵。

x = fminunc(fun,x0,options)：用 options 参数中指定的优化参数进行最小化。

x = fminunc(fun,x0,options,P1,P2,⋯)：将问题参数 P1、P2 等直接输给目标函数 fun，将 options 参数设置为空矩阵，作为 options 参数的默认值。

[x,fval] = fminunc(⋯)：将解 x 处目标函数的值返回到 fval 参数中。

[x,fval,exitflag] = fminunc(⋯)：返回 exitflag 值，描述函数的输出条件。

[x,fval,exitflag,output] = fminunc(⋯)：返回包含优化信息的结构输出。

[x,fval,exitflag,output,grad] = fminunc(⋯)：将解 x 处 fun 函数的梯度值返回到 grad 参数中。

[x,fval,exitflag,output,grad,hessian] = fminunc(⋯)：将解 x 处目标函数的 Hessian 矩阵信息返回到 hessian 参数中。

输入/输出变量的描述如表 6-1 所示。

表 6-1　输入/输出变量描述

变量	描　述
fun	目标函数,需要最小化的目标函数。fun 函数需要输入标量参数 x,返回 x 处的目标函数标量值 f。可以将 fun 函数指定为命令行,例如: x = fminbnd(inline('sin(x * x) '),x0) 同样,fun 函数可以是一个包含函数名的字符串。对应的函数可以是 M 文件、内部函数或 MEX 文件。若 fun='myfun',则 M 文件函数 myfun.m 必须有下面的形式: function f = myfun(x) f = … 若 fun 函数的梯度可以算得,且 options.GradObj 设为 on: options = optimset('GradObj', 'on') 则 fun 函数必须返回解 x 处的梯度向量 g 到第二个输出变量中去。当被调用的 fun 函数只需要一个输出变量时(如算法只需要目标函数的值而不需要其梯度值时),可以通过核对 nargout 的值来避免计算梯度值。其调用格式如下: function [f,g] = myfun(x) f=…:计算 x 处的函数值。 if nargout > 1:调用 fun 函数并要求有两个输出变量。 g=…:计算 x 处的梯度值。 end 若 Hessian 矩阵也可以求得,并且 options.Hessian 设为 on,即 options = optimset('Hessian', 'on') 则 fun 函数必须返回解 x 处的 Hessian 对称矩阵 H 到第 3 个输出变量中去。当被调用的 fun 函数只需要一个或两个输出变量时(如算法只需要目标函数的值 f 和梯度值 g 而不需要 Hessian 矩阵 H 时),可以通过核对 nargout 的值来避免计算 Hessian 矩阵
options	优化参数选项。可以通过 optimset 函数设置或改变这些参数。其中有的参数适用于所有的优化算法,有的则只适用于大型优化问题,另外一些则只适用于中型问题。首先描述适用于大型问题的选项。这仅是一个参考,因为使用大型问题算法需要满足一些条件。对于 fminunc 函数来说,必须提供梯度信息。 LargeScale:当设为 on 时,使用大型算法;若设为 off 时,则使用中型问题的算法。 适用于大型和中型算法的参数如下: 　　Diagnostics:打印最小化函数的诊断信息。 Display:显示水平。选择 off,不显示输出;选择 iter,显示每一步迭代过程的输出;选择 final,显示最终结果,打印最小化函数的诊断信息。 GradObj:用户定义的目标函数的梯度。对于大型问题此参数是必选的,对于中型问题则是可选项。 MaxFunEvals:函数评价的最大次数。 MaxIter:最大允许迭代次数。 TolFun:函数值的终止容限。 TolX:x 处的终止容限。 只用于大型算法的参数如下: Hessian:用户定义的目标函数的 Hessian 矩阵。 HessPattern:用于有限差分的 Hessian 矩阵的稀疏形式。若不方便求 fun 函数的稀疏 Hessian 矩阵 H,可以通过用梯度的有限差分获得的 H 的稀疏结构(如非零值的位置等)来得到近似的 Hessian 矩阵 H。若连矩阵的稀疏结构都不知道,则可以将 HessPattern 设为密集矩阵,在每一次迭代过程中,都将进行密集矩阵的有限差分近似(这是默认设置的)。这将非常麻烦,所以花一些力气得到 Hessian 矩阵的稀疏结构还是值得的。

续表

变量	描　　述
options	MaxPCGIter：PCG 迭代的最大次数。 PrecondBandWidth：PCG 前处理的上带宽，默认为零。对于有些问题，增加带宽可以减少迭代次数。 TolPCG：PCG 迭代的终止容限。 TypicalX：典型 x 值。 只用于中型算法的参数如下： DerivativeCheck：对用户提供的导数和有限差分求出的导数进行对比。 DiffMaxChange：变量有限差分梯度的最大变化。 DiffMinChange：变量有限差分梯度的最小变化。 LineSearchType：一维搜索算法的选择
exitflag	描述退出条件如下： ＞0：表示目标函数收敛于解 x 处。 0：表示已经达到函数评价或迭代的最大次数。 ＜0：表示目标函数不收敛
output	该参数包含的优化信息如下： output.iterations：迭代次数。 output.algorithm：所采用的算法。 output.funcCount：函数评价次数。 output.cgiterations：PCG 迭代次数(只适用于大型规划问题)。 output.stepsize：最终步长的大小(只用于中型问题)。 output.firstorderopt：一阶优化的度量，解 x 处梯度的范数

2. fminsearch 函数

该函数是求解多变量无约束函数的最小值，其调用格式如下：

fminsearch 求解多变量无约束函数的最小值。该函数常用于无约束非线性最优化问题。

x = fminsearch(fun,x0)：初值为 x0，求 fun 函数的局部极小点 x。x0 可以是标量、向量或矩阵。

x = fminsearch(fun,x0,options)：用 options 参数指定的优化参数进行最小化。

x = fminsearch(fun,x0,options,P1,P2,…)：将问题参数 P1、P2 等直接输给目标函数 fun，将 options 参数设置为空矩阵，作为 options 参数的默认值。

[x,fval] = fminsearch(…)：将 x 处的目标函数值返回到 fval 参数中。

[x,fval,exitflag] = fminsearch(…)：返回 exitflag 值，描述函数的退出条件。

[x,fval,exitflag,output] = fminsearch(…)：返回包含优化信息的输出参数 output。

3. fminbnd 函数

该函数是求解局部极小值点，只可能返回一个极小值点，其调用格式如下：

```
[x,fval] = fminbnd(fun,x1,x2,options)
x = fminbnd( … )
```

【例 6-4】 求 $\min f(x)=e^{-x}+2x^2$，其搜索区间为(0,2)。

解：编写 MATLAB 代码如下：

```
clear all
clc
[x,fval] = fminbnd('exp( - x) + 2 * x.^2',0,2)
```

运行后得到结果如下：

```
x =
    0.2039
fval =
    0.8987
```

4. lsqnonlin 函数

该函数为求解多元非线性最小二乘问题。

非线性最小二乘问题的数学模型为

$$\min f(x) = \sum_{i=1}^{m} f_i(x)^2 + L$$

其中 L 为常数。

函数调用格式如下：

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,options)
x = lsqnonlin(fun,x0,options,P1,P2)
[x,resnorm] = lsqnonlin( … )
[x,resnorm, residual,exitflag] = lsqnonlin( … )
[x,resnorm, residual , exitflag,output] = lsqnonlin( … )
[x,resnorm, residual,exitflag, output,lambda] = lsqnonlin( … )
[x,resnorm, r esidual,exitflag, output,lambda,jacobian] = lsqnonlin( … )
```

其中，x 返回解向量；resnorm 返回 x 处残差的平方范数值 sum(fun(x).^2)；residual 返回 x 处的残差值 fun(x)；lambda 返回包含 x 处 Lagrange 乘子的结构参数；jacobian 返回解 x 处的 fun 函数的雅可比矩阵。

lsqnonlin 默认时选择大型优化算法。Lsqnonlin 通过将 options. LargeScale 设置为'off'来选择中型优化算法，其采用一维搜索法。

【例 6-5】 求解 $\min f(x)=3(x_2-x_1)^2+(x_2-5)^2$，其初始点为(1,2)。

解：编写 MATLAB 程序如下：

```
clear all
clc
f = '3 * (x(2) - x(1))^2 + (x(2) - 5)^2'
[x,reshorm] = lsqnonlin(f,[1,2])
```

运行后得到结果如下：

```
Optimization completed because the size of the gradient is less than
1e-4 times the default value of the function tolerance.

<stopping criteria details>

x =
    4.9986   4.9990
reshorm =
   1.8986e-12
```

6.2.3 无约束非线性规划问题的应用

【例 6-6】 求解 $f(x)=5x_1^2+3x_1x_2+x_2^2$ 无约束非线性函数的最小值，其初始点为(1,2)。

解：编写 MATLAB 代码如下：

```
clear all
clc
x0 = [1,2];
[x,fval] = fminunc(@fun1,x0)

function f = fun1(x)
f = 5 * x(1)^2 + 3 * x(1) * x(2) + x(2)^2;
```

运行后得到结果如下：

```
Optimization completed because the size of the gradient is less than
the default value of the optimality tolerance.

<stopping criteria details>
x =
   1.0e-05 *
    0.1101   -0.1489
fval =
   3.3630e-12
```

【例 6-7】 求解 $f(x)=x_1^2-2x_1x_2-6x_2^2$ 无约束非线性函数的最小值，其初始点为(−1,1)。

解：编写 MATLAB 代码如下：

```
clear all
clc
x0 = [ - 1,1];
[x,fval] = fminunc(@fun1,x0)

function f = fun1(x)
f = x(1)^2 - 2 * x(1) * x(2) - 6 * x(2)^2;
```

运行后得到结果如下：

```
fminunc stopped because the objective function value is less than
or equal to the default value of the objective function limit.

<stopping criteria details>

x =
    1.0e+09 *
    1.5691   3.9226
fval =
  - 1.0217e+20
```

6.3 求解非线性规划

6.3.1 一维最优化方法

一维最优化方法是指寻求一元函数在某区间上最优值点的方法。这类方法不仅有实用价值，而且大量多维最优化方法都依赖于一系列的一维最优化方法。常用的一维最优化方法有黄金分割法、切线法和插值法。

(1) 黄金分割法，又称 0.618 法。它适用于单峰函数。其基本思想是在初始寻查区间中设计一列点，通过逐次比较其函数值，逐步缩小寻查区间，以得出近似最优值点。

(2) 切线法，又称牛顿法。它也是针对单峰函数的。其基本思想是在一个猜测点附近将目标函数的导函数线性化，用此线性函数的零点作为新的猜测点，逐步迭代去逼近最优点。

(3) 插值法，又称多项式逼近法。其基本思想是用多项式(通常用二次或三次多项式)去拟合目标函数。

此外，还有斐波那契法、割线法、有理插值法、分批搜索法等。

6.3.2 无约束最优化方法

无约束最优化方法是指上述一般非线性规划模型的求解方法。常用的约束最优化方法有以下 4 种：

(1) 拉格朗日乘子法：它是将原问题转化为求拉格朗日函数的驻点。

（2）制约函数法：又称系列无约束最小化法，简称 SUMT 法。它又分为两类，一类叫惩罚函数法，又称外点法；另一类叫障碍函数法，又称内点法。它们都是将原问题转化为一系列无约束问题来求解。

（3）可行方向法：这是一类通过逐次选取可行下降方向去逼近最优点的迭代算法。如佐坦迪克法、弗兰克-沃尔夫法、投影梯度法和简约梯度法都属于此类算法。

（4）近似型算法：这类算法包括序贯线性规划法和序贯二次规划法。前者将原问题化为一系列线性规划问题来求解，后者将原问题化为一系列二次规划问题来求解。

6.3.3 约束最优化方法

约束最优化方法是指寻求 n 元实函数在整个 n 维向量空间上的最优值点的方法。这类方法的意义在于虽然实用规划问题大多是有约束的，但许多约束最优化方法可将有约束问题转化为若干无约束问题来求解。

无约束最优化方法大多是逐次一维搜索的迭代算法。这类迭代算法可分为两类。一类需要用目标函数的导函数，称为解析法。另一类不涉及导数，只用到函数值，称为直接法。这些迭代算法的基本思想是在一个近似点处选定一个有利搜索方向，沿这个方向进行一维寻查，得出新的近似点。然后对新点施行同样手续，如此反复迭代，直到满足预定的精度要求为止。根据搜索方向的取法不同，可以有各种算法。

属于解析型的算法有以下 4 种：

（1）梯度法：又称最速下降法。这是早期的解析法，收敛速度较慢。

（2）牛顿法：收敛速度快，但不稳定，计算也较困难。

（3）共轭梯度法：收敛较快，效果较好。

（4）变尺度法：这是一类效率较高的方法。其中达维登-弗莱彻-鲍威尔变尺度法，简称 DFP 法，是最常用的方法。

属于直接型的算法有交替方向法（又称坐标轮换法）、模式搜索法、旋转方向法、鲍威尔共轭方向法和单纯形加速法等。

6.4 非线性规划实例

6.4.1 遗传算法求解非线性规划

通常非线性整数规划是一个具有指数复杂度的 NP 问题。如果约束较为复杂，MATLAB 优化工具箱和一些优化软件如 lingo 等，常常无法应用，即使能应用也不能给出一个较为令人满意的解。这时就需要针对问题设计专门的优化算法。

将遗传算法应用于非线性规划，是提高最优化质量改善收敛效果的有效途径。本节介绍遗传算法在非线性规划中的具体应用，设计并实现求解非线性规划问题的遗传算法。

【例 6-8】 标准遗传算法的一个重要概念是染色体是可能解的二进制顺序号，由这个序号在可能解的集合（解空间）中找到可能解。运用遗传算法求解设定的非线性方

程组。

分析：这是遗传算法由染色体(可能解的二进制)顺序号找到可能解，把解代入设定的非线性方程组计算误差函数，判定方程组是否有解函数，选择最优染色体函数。

解：遗传算法程序的流程如下：

(1) 程序初始化，随机生成一组可能解(第一批染色体)；

(2) 由可能解的序号寻找解本身；

(3) 把解代入非线性方程计算误差，如果误差符合要求，停止计算；

(4) 选择最优解对应的最优染色体；

(5) 保留每次迭代产生的最优染色体，以防最优染色体丢失；

(6) 把保留的最优染色体 holdBestChromosome 加入到染色体群中；

(7) 为每一条染色体(即可能解的序号)定义一个概率；

(8) 按照概率筛选染色体；

(9) 染色体杂交；

(10) 变异。

编写 MATLAB 代码如下：

```
%主函数%
clear all
clc
circleN = 200;                          %迭代次数
format long

%%%%%%%%构造可能解的空间,确定染色体的个数、长度%%%%%%%
solutionSum = 4;
leftBoundary = -10;
rightBoundary = 10;
distance = 1;
chromosomeSum = 500;                    %染色体的个数
solutionSumError = 0.1;                 %解的误差

oneDimensionSet = leftBoundary:distance:rightBoundary;

oneDimensionSetN = size(oneDimensionSet,2); %返回 oneDimensionSet 中的元素个数
solutionN = oneDimensionSetN ^ solutionSum;  %解空间(解集合)中可能解的总数
binSolutionN = dec2bin(solutionN);           %把可能解的总数转换成二进制数
chromosomeLength = size(binSolutionN,2);     %由解空间中可能解的总数(二进制数)计算染色
                                              体的长度

%%%%%%%%%%%%%%%%程序初始化%%%%%%%%%%%%%%%%%%%%
solutionSequence = fix(rand(chromosomeSum,1) * solutionN) + 1;
for i = 1:chromosomeSum
if solutionSequence(i)> solutionN;
solutionSequence(i) = solutionN;
end
end
```

```
%染色体是解集合中的序号,它对应一个可能解
%把解的十进制序号转成二进制序号
fatherChromosomeGroup = dec2bin(solutionSequence,chromosomeLength);
holdLeastFunctionError = Inf;                        %可能解的最小误差的初值
holdBestChromosome = 0;                              %对应最小误差的染色体的初值

%%%%%%% 开始计算 %%%%%%%%%%%%
circle = 0;
while circle < circleN %开始迭代求解
circle = circle + 1;                                 %记录迭代次数
%%%%%由可能解的序号寻找解本身%%%%%%%%
x = chromosome_x(fatherChromosomeGroup,oneDimensionSet,solutionSum);
%%%%%把解代入非线性方程计算误差%%%%%%%%
functionError = nonLinearSumError1(x);               %把解代入方程计算误差
[solution,minError,isTrue] = isSolution(x,functionError,solutionSumError);
if isTrue == 1
'方程得解'
solution
minError
circle
return %结束程序
end
%%%%%选择最优解对应的最优染色体%%%%%%%%
[ bestChromosome, leastFunctionError ] = best _ worstChromosome ( fatherChromosomeGroup,
functionError);
%%%%%%保留每次迭代产生的最优染色体%%%%%%%
[holdBestChromosome,holdLeastFunctionError]...
 = compareBestChromosome(holdBestChromosome,holdLeastFunctionError,...
bestChromosome,leastFunctionError);

%%%%%%把保留的最优染色体加入到染色体群中%%%%%%%%
order = round(rand(1) * chromosomeSum);
if order == 0
order = 1;
end
fatherChromosomeGroup(order,:) = holdBestChromosome;
functionError(order) = holdLeastFunctionError;

%%%%%%为每一条染色体,即可能解的序号定义一个概率%%%%%%%%%
[p,trueP] = chromosomeProbability(functionError);
if trueP == 'Fail'
'可能解严重不适应方程,请重新开始'
return %结束程序
end
%%%%%%%%%%按照概率筛选染色体%%%%%%
fa = bin2dec(fatherChromosomeGroup);                 %显示父染色体
%从父染体中选择优秀染色体
selecteChromosomeGroup = selecteChromosome(fatherChromosomeGroup,p);
```

```
%%%%%%%% 染色体杂交 %%%%%%%
sle = bin2dec(selecteChromosomeGroup);
sonChromosomeGroup = crossChromosome(selecteChromosomeGroup,2);
sonChromosomeGroup = crossChromosome(fatherChromosomeGroup,2);
sonChromosomeGroup = checkSequence(sonChromosomeGroup,solutionN);
%检查杂交后的染色体是否越界
%%%%%%%% 杂交后变异 %%%%%%%
fatherChromosomeGroup = varianceCh(sonChromosomeGroup,0.1,solutionN);
fatherChromosomeGroup = checkSequence(fatherChromosomeGroup,solutionN);%检查变异后的染
                                                                      色体是否越界
end


%由染色体(可能解的二进制)顺序号找到可能解
%这个函数找出染色体(可能解的序号)对应的可能解 x
function x = chromosome_x(chromosomeGroup,oneDimensionSet,solutionSum)
%chromosomeGroup:染色体,也是可能解的二进制序号
%oneDimensionSet:一维数轴上的可能解
%solutionSum:非线性方程组的元数,也就是待解方程中未知变量的个数
[row oneDimensionSetN] = size(oneDimensionSet);

chromosomeSum = size(chromosomeGroup);   %chromosomeSum:染色体的个数
xSequence = bin2dec(chromosomeGroup);    %把可能解的二进制序号(染色体)转换成十进制序号
for i = 1:chromosomeSum                  %i:染色体的编号
remainder = xSequence(i);
for j = 1:solutionSum
dProduct = oneDimensionSetN^(solutionSum - j);        %sNproduct:
quotient = remainder/dProduct;
remainder = mod(remainder,dProduct);                  %mod:取余函数
if remainder == 0
oneDimensionSetOrder = quotient;
else
oneDimensionSetOrder = fix(quotient) + 1;             %fix:取整函数
end
if oneDimensionSetOrder == 0
oneDimensionSetOrder = oneDimensionSetN;
end
x(i,j) = oneDimensionSet(oneDimensionSetOrder);
end
end

%把解代入非线性方程组计算绝对误差函数
function funtionError = nonLinearSumError1(X)%方程的解是 -7,5,1,-3
funtionError = ...
[
abs(X(:,1).^2 - sin(X(:,2).^3) + X(:,3).^2 - exp(X(:,4)) - 50.566253390821) + ...
abs(X(:,1).^3 + X(:,2).^2 - X(:,4).^2 + 327) + ...
abs(cos(X(:,1).^4) + X(:,2).^4 - X(:,3).^3 - 624.679868769613) + ...
abs(X(:,1).^4 - X(:,2).^3 + 2.^X(:,3) - X(:,4).^4 - 2197)
```

```
];

% 判定程序是否得解函数
function [solution,minError,isTrue] = isSolution(x,functionError,precision)
[minError,xi] = min(functionError);        % 找到最小误差,最小误差所对应的行号
solution = x(xi,:);
if minError < precision
isTrue = 1;
else
isTrue = 0;
end

% 选择最优染色体函数
% 找出最小误差所对应的最优染色体,最大误差所对应的最坏染色体
function [bestChromosome, leastFunctionError] = best_worstChromosome(chromosomeGroup,
functionError)
[leastFunctionError minErrorOrder] = min(functionError);

bestChromosome = chromosomeGroup(minErrorOrder,:);


% 误差比较函数:从两个染色体中选出误差较小的染色体保留下来
function [newBestChromosome,newLeastFunctionError]...
= compareBestChromosome(oldBestChromosome,oldLeastFunctionError,...
bestChromosome,leastFunctionError)
if oldLeastFunctionError > leastFunctionError
newLeastFunctionError = leastFunctionError;
newBestChromosome = bestChromosome;
else
newLeastFunctionError = oldLeastFunctionError;
newBestChromosome = oldBestChromosome;
end

% 为染色体定义概率函数,好染色体概率高,坏染色体概率低
% 根据待解的非线性函数的误差计算染色体的概率
function [p,isP] = chromosomeProbability(x_Error)
InfN = sum(isinf(x_Error));                % 估计非线性方程计算的结果
NaNN = sum(isnan(x_Error));
if InfN > 0 || NaNN > 0
isP = 'Fail';
p = 0;
return
else
isP = 'True';
errorReciprocal = 1./x_Error;
sumReciprocal = sum(errorReciprocal);
p = errorReciprocal/sumReciprocal;         % p:可能解所对应的染色体的概率
end
```

```
%按概率选择染色体函数
function chromosome = selecteChromosome(chromosomeGroup,p)
cumuP = cumsum(p);                %累积概率,也就是把每个染色体的概率映射到0~1的区间
[chromosomeSum,chromosomeLength] = size(chromosomeGroup);
for i = 1:chromosomeSum           %这个循环产生概率值
rN = rand(1);
if rN == 1
chromosome(i,:) = chromosomeGroup(chromosomeSum,:);
elseif (0 <= rN) && (rN < cumuP(1))
chromosome(i,:) = chromosomeGroup(1,:);     %第1条染色体被选中
else
for j = 2:chromosomeSum           %这个循环确定第1条以后的哪一条染色体被选中
if (cumuP(j - 1)<= rN) && (rN < cumuP(j))
chromosome(i,:) = chromosomeGroup(j,:);
break
end
end
end
end

%父代染色体杂交产生子代染色体函数
function sonChromosome = crossChromosome(fatherChromosome,parameter)
[chromosomeSum,chromosomeLength] = size(fatherChromosome);
%chromosomeSum:染色体的条数; chromosomeLength: 染色体的长度
switch parameter
case 1                            %随机选择父染色体进行交叉重组
for i = 1:chromosomeSum/2
crossDot = fix(rand(1) * chromosomeLength);    %随机选择染色体的交叉点位
randChromosomeSequence1 = round(rand(1) * chromosomeSum);
%随机产生第1条染色体的序号
randChromosomeSequence2 = round(rand(1) * chromosomeSum);
%随机产生第2条染色体的序号,这两条染色体要进行杂交
if randChromosomeSequence1 == 0                %防止产生0序号
randChromosomeSequence1 = 1;
end
if randChromosomeSequence2 == 0                %防止产生0序号
randChromosomeSequence2 = 1;
end
if crossDot == 0 || crossDot == 1
sonChromosome(i * 2 - 1,:) = fatherChromosome(randChromosomeSequence1,:);
sonChromosome(i * 2,:) = fatherChromosome(randChromosomeSequence2,:);
else
%执行两条染色体的交叉
sonChromosome(i * 2 - 1,:) = fatherChromosome(randChromosomeSequence1,:);
%把父染色体整条传给子染色体
sonChromosome(i * 2 - 1,crossDot:chromosomeLength) = ...
fatherChromosome(randChromosomeSequence2,crossDot:chromosomeLength)
%下一条父染色体上交叉点crossDot后的基因传给子染色体,完成前一条染色体的交叉
sonChromosome(i * 2,:) = fatherChromosome(randChromosomeSequence2,:);
sonChromosome(i * 2,crossDot:chromosomeLength)...
```

```
 = fatherChromosome(randChromosomeSequence1,crossDot:chromosomeLength)
end
end
case 2                              % 父染色体的第 i 号与第 chromosomeSum + 1 - i 号交叉
for i = 1:chromosomeSum/2
crossDot = fix(rand(1) * chromosomeLength);          % 随机选择染色体的交叉点位
if crossDot == 0 || crossDot == 1
sonChromosome(i * 2 - 1,:) = fatherChromosome(i,:);
sonChromosome(i * 2,:) = fatherChromosome(chromosomeSum + 1 - i,:);
else
 % 执行两条染色体的交叉
sonChromosome(i * 2 - 1,:) = fatherChromosome(i,:);  % 把父染色体整条传给子染色体
sonChromosome(i * 2 - 1,crossDot:chromosomeLength)...
 = fatherChromosome(chromosomeSum + 1 - i,crossDot:chromosomeLength);
 % 下一条父染色体上交叉点 crossDot 后的基因传给子染色体,完成前一条染色体的交叉
sonChromosome(i * 2,:) = fatherChromosome(chromosomeSum + 1 - i,:);
sonChromosome(i * 2,crossDot:chromosomeLength)...
 = fatherChromosome(i,crossDot:chromosomeLength);
end
end
case 3                              % 父染色体的第 i 号与第 i + chromosomeSum/2 号交叉
for i = 1:chromosomeSum/2
crossDot = fix(rand(1) * chromosomeLength);          % 随机选择染色体的交叉点位
if crossDot == 0 || crossDot == 1
sonChromosome(i * 2 - 1,:) = fatherChromosome(i,:);
sonChromosome(i * 2,:) = fatherChromosome(i + chromosomeSum/2,:);
else
 % 执行两条染色体的交叉
sonChromosome(i * 2 - 1,:) = fatherChromosome(i,:);  % 把父染色体整条传给子染色体
sonChromosome(i * 2 - 1,crossDot:chromosomeLength)...
 = fatherChromosome(i + chromosomeSum/2,crossDot:chromosomeLength);
 % 下一条父染色体上交叉点 crossDot 后的基因传给子染色体,完成前一条染色体的交叉
sonChromosome(i * 2,:) = fatherChromosome(i + chromosomeSum/2,:);
sonChromosome(i * 2,crossDot:chromosomeLength)...
 = fatherChromosome(i,crossDot:chromosomeLength);
end
end
end


 % 防止染色体超出解空间的函数
 % 检测染色体(序号)是否超出解空间的函数
function chromosome = checkSequence(chromosomeGroup,solutionSum)
[chromosomeSum,chromosomeLength] = size(chromosomeGroup);
decimalChromosomeSequence = bin2dec(chromosomeGroup);
for i = 1:chromosomeSum                              % 检测变异后的染色体是否超出解空间
if decimalChromosomeSequence(i)> solutionSum
chRs = round(rand(1) * solutionSum);
if chRs == 0
chRs = 1;
```

```
end
decimalChromosomeSequence(i) = chRs;
end
end
chromosome = dec2bin(decimalChromosomeSequence,chromosomeLength);


%变异函数
%基因变异.染色体群中的1/10变异.vR是变异概率.solutionN是解空间中全部可能解的个数
function aberranceChromosomeGroup = varianceCh(chromosomeGroup,vR,solutionN)
[chromosomeSum,chromosomeLength] = size(chromosomeGroup);
if chromosomeSum < 10
N = 1;
else
N = round(chromosomeSum/10);
end

if rand(1)> vR                                              %变异操作
for i = 1:N
chromosomeOrder = round(rand(1) * chromosomeSum);           %产生变异染色体序号
if chromosomeOrder == 0
chromosomeOrder = 1;
end
aberrancePosition = round(rand(1) * chromosomeLength);      %产生变异位置
if aberrancePosition == 0
aberrancePosition = 1;
end
if chromosomeGroup(chromosomeOrder,aberrancePosition) == '1'
chromosomeGroup(chromosomeOrder,aberrancePosition) = '0';   %变异
else
chromosomeGroup(chromosomeOrder,aberrancePosition) = '1';   %变异
end
end
aberranceChromosomeGroup = chromosomeGroup;
else
aberranceChromosomeGroup = chromosomeGroup;
end
```

运行后得到结果如下：

```
ans =
方程得解
solution =
    -7  5  1  -3
minError =
     5.471179065352771e-13
circle =
   166
```

即迭代 166 次，方程的解为[－7，5，1，－3]，解的误差为 5.471179065352771e－13。

6.4.2 资金调用问题

【例 6-9】 假设某公司有 400 万元资金，要求在 4 年内使用完。若在一年内使用资金 x 万元，则可获得效益$\sqrt{x}$万元（设效益不再投资），当年不用的资金可存入银行，年利率为 10%。试制定出这笔资金的使用方案，以使 4 年的经济效益总和达到最大。

分析：针对现有资金 400 万元，对于不同的使用方案，4 年内所获得的效益总和是不相同的。比如第一年就把 400 万元全部用完，这获得的效益总和为 $\sqrt{400}=20.0$ 万元。若前三年均不用这笔资金，而把它存入银行，则第四年时的本息和为 $400\times1.1^3=532.4$ 万元，再把它全部用完，则效益总和为 23.07 万元，比第一种方案效益多 3 万多元。所以用最优化方法可以制定出一种最优的使用方案，以使 4 年的经济效益总和达到最大。

解：设 x_i 表示第 i 年所使用资金数，T 表示 4 年的效益总和，则目标函数为

$$\max T=\sqrt{x_1}+\sqrt{x_2}+\sqrt{x_3}+\sqrt{x_4}$$

决策变量的约束条件：每一年所使用资金既不能为负数，也不能超过当年所拥有的资金数，即第一年使用的资金数 x_1，满足 $0\leqslant x_1\leqslant 400$；第二年资金数 x_2，满足：$0\leqslant x_2\leqslant(400-x_1)\times1.1$（第一年未使用资金存入银行一年后的本利之和）；第三年资金数 x_3，满足 $0\leqslant x_3\leqslant[(400-x_1)\times1.1-x_2\times1.1$；第四年资金数 x_4，满足 $0\leqslant x_4\leqslant\{[(400-x_1)\times1.1-x_2]\times1.1-x_3\}\times1.1$。

这样，资金使用问题的数学模型为

$$\max T=\sqrt{x_1}+\sqrt{x_2}+\sqrt{x_3}+\sqrt{x_4}$$

$$\text{s.t.}\begin{cases}x_1\leqslant 400\\1.1x_1+x_2\leqslant 440\\1.21x_1+1.1x_2+x_3\leqslant 484\\1.331x_1+1.21x_2+1.1x_3+x_4\leqslant 532.4\\x_1,x_2,x_3,x_4\geqslant 0\end{cases}$$

模型的求解，这是非线性规划模型的求解问题，可选用的函数格式如下：

```
[x.fval] = fmincon(fun,x0,a,b,Aeq,beq,lb,ub)
```

首先，用极小化的形式将目标函数改写为

$$\min T=-\sqrt{x_1}-\sqrt{x_2}-\sqrt{x_3}-\sqrt{x_4}$$

其次，将约束条件表示为

$$\begin{cases}\boldsymbol{Ax}\leqslant\boldsymbol{b}\\\boldsymbol{lb}\leqslant\boldsymbol{x}\leqslant\boldsymbol{ub}\end{cases}$$

其中各输入参数为

$$\boldsymbol{x}=[x_1,x_2,x_3,x_4]^{\mathrm{T}},\quad \boldsymbol{lb}=[0,0,0,0]^{\mathrm{T}},\quad \boldsymbol{ub}=[400,1000,1000,1000]^{\mathrm{T}}$$

$$A = \begin{bmatrix} 1.1 & 1 & 0 & 0 \\ 1.21 & 1.1 & 1 & 0 \\ 1.331 & 1.21 & 1.1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 440 \\ 484 \\ 532.4 \end{bmatrix}$$

根据以上分析，首先编写目标函数

```
function y = zhangsan(x)
y = - sqrt(x(1)) - sqrt(x(2)) - sqrt(x(3)) - sqrt(x(4));
```

编写主程序：

```
clear all
clc
A = [1.1,1,0,0;1.21,1.1,1,0;1.331,1.21,1.1,1];
b = [440,484,532.4];
lb = [0,0,0,0];
ub = [400,1000,1000,1000];
x0 = [100,100,100,100];
[x,fval] = fmincon('mubiao',x0,A,b,[],[],lb,ub)
```

运行后得到结果如下：

```
x =

   1.0e + 02 *

   0.861881539970750   1.042879152818849   1.261877085155912   1.526887081944790


fval =

 - 43.085960510340023
```

即 4 年使用资金数分别为 84.2440、107.6353、128.9031、148.2391 万元，使得 4 年收益总和为 43.0821 万元。

【例 6-10】 设有 100 万元资金，要求 5 年内使用完，若在一年内使用资金 x 万元，则可得效益 $\sqrt{2x}$ 万元（效益不能再使用），当年不用的资金可存入银行，年利率为 5%。试制定出资金的使用计划，使 5 年效益之和为最大。

解： 设变量 x_i 表示第 i 年所使用的资金数 z，则有

$$\max z = \sqrt{2x_1} + \sqrt{2x_2} + \sqrt{2x_3} + \sqrt{2x_4} + \sqrt{2x_5}$$

$$x_1 \leqslant 100$$

$$1.05x_1 + x_2 \leqslant 105$$

$$1.1025x_1 + 1.05x_2 + x_3 \leqslant 110.25$$

$$1.1576x_1 + 1.1025x_2 + 1.05x_3 + x_4 \leqslant 115.76$$

$$1.2155x_1 + 1.1576x_2 + 1.1025x_3 + 1.05x_4 + x_5 \leqslant 121.55$$

$$x_i \geqslant 0, i = 1,2,3,4,5$$

根据以上编写 MATLAB 程序如下：

```
%%% 函数 fun %%%
function f = fun(x)
f = - (sqrt(2 * x(1)) + sqrt(2 * x(2)) + sqrt(2 * x(3)) + sqrt(2 * x(4))  + sqrt(2 * x(5)));

%%% 函数 mycon %%%
function [g,ceq] = mycon(x)
g(1) = x(1) - 100;
g(2) = 1.05 * x(1) + x(2) - 105;
g(3) = 1.1025 * x(1) + 1.05 * x(2) + x(3) - 110.25;
g(4) = 1.1576 * x(1) +  1.1025 * x(2) + 1.05 * x(3) + x(4) - 115.76;
g(5) = 1.331 * x(1) +  1.1576 * x(2) +  1.1025 * x(3) + 1.05 * x(4) + x(5) - 121.55;

ceq = 0

%%% 主函数 mycon %%%
clear all
clc
x0 = [1;1;1;1;1];
vlb = [0;0;0;0;0];
vub = [];
A = [];
b = [];
Aeq = [];
beq = [];
[x,fval] = fmincon('fun',x0,A,b,Aeq,beq,vlb,vub,'mycon')
```

运行结果为

```
x =

   15.3337
   20.2715
   22.3484
   24.6390
   27.1646


fval =

  - 32.9814
```

即表示 x_1=15.3337；x_2=20.2715；x_3=22.3484；x_4=24.6390；z=27.1646。

6.4.3 经营最佳安排问题

【例 6-11】 假设某公司经营两种设备，第一种设备每件售价 30 元，第二种设备每件售价 450 元，根据统计售出第一件第一种设备所需的营业时间平均为 0.5 小时，第二种设备是$(2+0.25\times x_2)$小时，其中 x_2 是第二种设备的售出数量，已知该公司在这段时间内的总营业时间为 800 小时，试确定使营业额最大的营业计划。

分析：设该公司计划经营第一种设备 x_1 件，第二种设备 x_2 件。根据题意，建立如下的数学模型：

$$\max f(x) = 30x_1 + 450x_2$$

$$\text{s. t.}\begin{cases}0.5x_1 + (2+0.25x_2)x_2 = 800 \\ x_1, x_2 \geqslant 0\end{cases}$$

解：根据分析编写 MATLAB 代码如下：

```
%M文件来定义目标函数%
function f = mubiao(x)
f = - 30 * x(1) - 450 * x(2);

%编写约束条件的M文件%
function [c.cep] = yuechu(x)
c = 0.5 * x(1) + 2 * x(2) + 0.25 * x(2) * x(2) - 800;
cep = [];

%主程序%
clear all
clc
lb = [0,0];
x0 = [0,0];
[x,w] = fmincon('mubiao',x0,[],[],[],[],lb,[],'yuechu')
```

运行得到的结果如下：

```
x =
   1.0e+03 *
   1.495499926605988   0.011000004053403
w =
     - 4.981499962221115e+04
```

即该公司经营第一种设备 1496 件，经营第二种设备 11 件，可使总营业额最大为 49815 元。

本章小结

非线性规划是20世纪50年代才开始形成的一门新兴学科。70年代又得到进一步的发展。非线性规划在工程、管理、经济、科研、军事等方面都有广泛的应用，为最优设计提供了有力的工具。

本章首先介绍了非线性规划基础，包括其标准形式和MATLAB函数。随后介绍了无约束非线性规划和非线性规划的求解，包括有约束和无约束最优方法。最后举例介绍了非线性规划的应用求解。

第7章 无约束一维极值

在求解目标函数的极小值的过程中，若对设计变量的取值范围不加限制，称这种最优化问题为无约束优化问题。尽管对于机械的优化设计问题，多数是有约束的，但无约束最优化方法仍然是最优化设计的基本组成部分。因为约束最优化问题可以通过对约束条件的处理，转化为无约束最优化问题来求解。

无约束一维极值问题求解时一般采用一维搜索法，其中方法包括多种。线性搜索包括黄金分割、斐波那契法、牛顿法等，非线性搜索包括抛物线法和三次插值法。本章主要介绍了求解无约束一维极值问题的不同方法。

学习目标：

（1）了解无约束算法基础；

（2）掌握 MATLAB 中的进退算法、黄金分割法等运算。

7.1 无约束算法基础

无约束最优化算法求解无约束最优化问题的方法，有解析法和直接法两类：

（1）解析法就是利用无约束最优化问题中目标函数 $f(x)$ 的解析表达式和它的解析性质（如函数的一阶导数和二阶导数），给出一种求它的最优解的方法，或一种求近似解的迭代方法。解析法主要有最速下降法、共轭方向法、共轭梯度法、非二次函数的共轭梯度法、牛顿法、拟牛顿法、变尺度法等。

（2）直接法就是在求最优解的过程中，只用到函数的函数值，而不必利用函数的解析性质。直接法也是一种迭代法，迭代步骤简单。当目标函数的表达式十分复杂，或写不出具体表达式时，它就成为了重要的方法。直接法适应面很广，适用于计算机运算。

无约束优化问题的一般形式可描述为

求 n 维随机变量

$$\boldsymbol{X} = [x_1, x_2, \cdots, x_n]^{\mathrm{T}} \in \mathbf{R}^n$$

使目标函数 $f(X)$ 最小。

对于上述无约束优化问题的求解，可以利用第 2 章讲述的无约束优化问题的极值条件求得，即将求目标函数的极值问题变成求方程 $\min f(X^*)=0$ 的解。也就是求 X^* 使其满足

$$\begin{cases} \dfrac{\partial f(X^*)}{\partial x_1} = 0 \\ \dfrac{\partial f(X^*)}{\partial x_2} = 0 \\ \cdots \\ \dfrac{\partial f(X^*)}{\partial x_n} = 0 \end{cases}$$

解上述方程组，求得驻点后，再根据极值点所需满足的充分条件来判定是否为极小值点。

但上式是一个含有 n 个未知量、n 个方程的方程组，而且在实际问题中一般是非线性的，很难用解析法求解，而是要用数值计算的方法。

优化问题的一般解法是数值迭代法。因此，与其用数值方法求解非线性方程组，还不如用数值迭代法直接求解无约束极值问题。

数值迭代法的基本思想是从一个初始点 $X^{(0)}$ 出发，按照一个可行的搜索方向 $\vec{d}^{(0)}$ 搜索，确定最佳的步长 α_0 使函数值沿 $\vec{d}^{(0)}$ 方向下降最大，得到 $X^{(1)}$ 点。依次一步一步地重复数值计算，最终达到最优点。优化计算所采用的基本迭代公式为

$$X^{(K+1)} = X^{(K)} + \alpha_K \vec{d}^{(0)(K)} \quad (K = 0,1,2,\cdots)$$

由上面的迭代公式可以看出，采用数值法进行迭代求优时，需要确定初始点 $X^{(k)}$、搜索方向 $\vec{d}^{(k)}$ 和迭代步长 α_K，这三项称为优化迭代算法的三要素。

一个算法的搜索方向是该优化方法的基本标志，它从根本上决定着一个算法的成败和收敛速率的快慢。因此，分析和确定搜索方向是研究优化方法的最根本任务之一。各种无约束优化方法也是在确定搜索方向上显示出各自的特点。

本章后续小节将分别介绍不同无约束一维极值优化方法。

7.2 进退法

进退法是一种缩小极值区间的算法，算出的结果是一个包含极值的区间，适用于未知极值范围的情况下使用。

其理论依据是 $f(x)$ 为单谷函数（只有一个极值点），且 $[a,b]$ 为其极小值点的一个搜索区间，对于任意 $x_1,x_2\in[a,b]$，如果 $f(x_1)<f(x_2)$，则 $[a,x_2]$ 为极小值的搜索区间，如果 $f(x_1)>f(x_2)$，则 $[x_1,b]$ 为极小值的搜索区间。

因此，在给定初始点 x_0 及初始搜索步长 h 的情况下，首先以初始步长向前搜索一步，计算 $f(x_0+h)$。

如果 $f(x_0)>f(x_0+h)$，则可知搜索区间为 $[x_0,\tilde{x}]$，其中 $\tilde{x}$ 待求，为确定 $\tilde{x}$，前进一步计算 $f(x_0+\lambda h)$，λ 为放大系数，且 $\lambda>1$，直到找到合适的 λ^*，使得 $f(x_0+h)<f(x_0+\lambda^* h)$，从

而确定搜索区间为$[x_0, x_0+\lambda^* h]$。

如果$f(x_0)<f(x_0+h)$，则可知搜索区间为$[\tilde{x}, x_0+h]$，其中$\tilde{x}$待求，为确定$\tilde{x}$，后退一步计算$f(x_0-\lambda h)$，λ为缩小系数，且$0<\lambda<1$，直到找到合适的λ^*，使得$f(x_0-\lambda^* h)>f(x_0)$，从而确定搜索区间为$[x_0-\lambda^* h, x_0+h]$。

进退法的基本算法步骤如下：

(1) 给定初始点$x^{(0)}$，初始步长h_0，令$h=h_0$，$x^{(1)}=x^{(0)}$，$k=0$；

(2) 令$x^{(4)}=x^{(1)}+h$，$k=k+1$；

(3) 若$f(x^{(4)})<f(x^{(1)})$，则转到步骤(4)，否则转到步骤(5)；

(4) 令$x^{(2)}=x^{(1)}$，$x^{(1)}=x^{(4)}$，$f(x^{(2)})=f(x^{(1)})$，$f(x^{(1)})=f(x^{(4)})$，，令$h=2h$，转到步骤(2)；

(5) 若$k=1$，则转到步骤(6)，否则转到步骤(7)；

(6) 令$h=-h$，$x^{(2)}=x^{(4)}$，$f(x^{(2)})=f(x^{(4)})$，转到步骤(2)；

(7) 令$x^{(3)}=x^{(2)}$，$x^{(2)}=x^{(1)}$，$x^{(1)}=x^{(4)}$，停止计算，极小值点包含于区间$[x^{(1)}, x^{(3)}]$或$[x^{(3)}, x^{(1)}]$。

根据以上分析，编写用进退法求解一维函数的极值区间的 MATLAB 函数 fun_JT 如下：

```
function [minx,maxx] = fun_JT(f,x0,h0,eps)
% 目标函数:f
% 初始点:x0
% 初始步长:h0
% 精度:eps
% 目标函数取包含极值的区间左端点:minx
% 目标函数取包含极值的区间又端点:maxx
format long;
if nargin == 3;eps = 1.0e-6;end
x1 = x0;k = 0;h = h0;
while 1
    x4 = x1 + h;              % ÊÔÌ½½
    k = k + 1;
    f4 = subs(f,findsym(f),x4);
    f1 = subs(f,findsym(f),x1);
    if f4 < f1
        x2 = x1;
        x1 = x4;
        f2 = f1;
        f1 = f4;
        h = 2 * h;            % ¼Ó²½³¤
    else
            if k == 1
                h = - h;    % ÏòºóËÑ÷
                x2 = x4;
                f2 = f4;
            else
                x3 = x2;
                x2 = x1;
```

```
                x1 = x4;
                break;
            end
    end
end
minx = min(x1,x3);
maxx = x1 + x3 - minx;
format short;
```

其调用格式为[min$x$,max$x$]=fun_JT(f,x0,h0,eps)

其中,f：目标函数；

x0：初始点；

h0：初始步长；

eps：精度；

minx：目标函数取包含极值的区间左端点；

maxx：目标函数取包含极值的区间右端点。

【例 7-1】 取初始点为 0,步长为 0.05,用进退法求函数 $f(t)=t^4-2t^2-t+1$ 的极值区间。

解：在 MATLAB 命令窗口中输入

```
clear all
clc
syms t;
f = t^4 - 2 * t^2 - t + 1;
[x1,x2] = fun_JT(f,0,0.05)
```

运行后得到结果如下：

```
x1 =

    0.3500


x2 =

    1.5500
```

由上面的结果可知 $f(t)$的极值点在区间[0.35,1.55]内。

【例 7-2】 用进退法求函数 $f(x)=x^2-10x-36$ 的极值。其中,$a=0.1$,$h=0.1$,$e=0.05$。

解：首先编写函数代码如下：

```
function y = f(x)
if nargin == 1
    y = x^2 - 10 * x - 36;
end
end
```

再根据进退法原理编写如下代码：

```
function y = jintui(a,h,e)
a = input('input the value of a:');
h = input('input the value of h:');
e = input('input the value of e:');
m = 1;
n = 1;
x0 = a;
x1 = x0 + h;
while abs(x1 - x0)> e
    if f(x0)> f(x1)
        x0 = x1;
        x1 = x0 + h * 2 ^m;
        m = m + 1;
    else
        x1 = x0;
        x0 = x1 - h/2 ^n;
        n = n + 1;
    end
end
x = (x0 + x1)/2;
y = f(x);
end
```

运行并分别输入参数值得到结果如下：

```
>> jintui
input the value of a:0.1
input the value of h:0.1
input the value of e:0.05

ans =

  - 59.1094
```

即函数极值为－59.1094。

7.3 黄金分割法

黄金分割法适用于已知极值区间的前提下，利用不断缩小区间的思想，最终得出极值的近似值。该方法只是要求函数单峰，可以不连续。因此，这种方法的适应面非常广泛。

黄金分割法也是建立在区间消去法原理基础上的试探方法，即在搜索区间$[a,b]$内适当插入两点a_1、a_2，并计算其函数值。a_1、a_2将原来区间分成三段，再应用函数的单峰性质，通过函数值大小的比较，删除其中一段，使搜索区间得以缩小。然后在保留下来的区间上作同样的处理，如此迭代下去，使搜索区间无限缩小，从而得到极小值点的数值近

似解。

黄金分割法是用于一元函数 $f(x)$ 在给定的初始区间 $[a,b]$ 内搜索极小值点 a^* 的一种方法。它是优化计算中的经典算法，以算法简单、收敛速度均匀、效果较好而著称，是许多优化算法的基础。但它只适用于一维区间上的凸函数，即只在单峰区间内才能进行一维寻优，其收敛效率较低。其基本原理是依照去劣存优原则、对称原则以及等比收缩原则来逐步缩小搜索区间。

黄金分割法的基本步骤如下：

(1) 给定区间 $[a,b]$ 及 $eps>0$；

(2) 计算 $r=a+0.382(b-a)$，$u=a+0.618(b-a)$；

(3) 若 $f(r)>f(u)$，则进行下一步，否则直接进行第(5)步；

(4) 若 $u-r<eps$，则停止计算，输出 $x^*=u$，$f^*=f(u)$，否则令 $a=r$，$r=u$，$u=a+0.618(b-a)$，转入第(3)步；

(5) 若 $u-r<eps$，则停止计算，输出 $x^*=r$，$f^*=f(r)$，否则令 $b=u$，$u=r$，$r=a+0.382(b-a)$，转入第(3)步。

根据以上步骤，编写黄金分割算法的MATLAB代码如下：

```
function [x,fval,iter] = HJ(a,b)
iter = 0;
while abs(b - a)> 1e - 5
    iter = iter + 1;
    lambda = a + 0.382 * (b - a);
    miu = a + 0.618 * (b - a);
    [dy,f1] = funHJ(lambda);
    [dy,f2] = funHJ(miu);
    if f1 > f2
        a = lambda;
        disp(['the' num2str(iter) 'iteration search area is :[' num2str(a) ',' num2str(b) ']'])
    else
        b = miu;
        disp(['the' num2str(iter) 'iteration search area is :[' num2str(a) ',' num2str(b) ']'])
    end
end
x = (a + b)/2;
[dy, fval] = funHJ(x);
```

【例 7-3】 根据黄金分割算法编写程序，求函数 $f(x)=(\sin x)^6[\tan(1-x)]e^{30x}$ 在区间 $[0,1]$ 上的极大值。

解： 令 $g(x)=-f(x)=-(\sin x)^6[\tan(1-x)]e^{30x}$，求 $\min\limits_{x\in[0,1]} g(x)$。

编写函数代码如下：

```
function [dy,val] = funHJ(x)
val = - ((sin(x))^6 * tan(1 - x) * exp(30 * x));
dy = - (6 * (sin(x))^5 * cos(x) * tan(1 - x) * exp(30 * x) + ...
    (sin(x))^6 * (1/cos(1 - x))^2 * ( - 1) * exp(30 * x) + (sin(x))^6 * tan(1 - x) * exp(30 *
x) * 30);
```

根据函数和黄金分割算法代码,求解代码如下:

```
clear all
clc
[x,fval,iter] = HJ(0,1)
```

运行后得到结果如下:

```
the1iteration search area is :[0.382,1]
the2iteration search area is :[0.61808,1]
the3iteration search area is :[0.76397,1]
the4iteration search area is :[0.85413,1]
the5iteration search area is :[0.90985,1]
the6iteration search area is :[0.94429,1]
the7iteration search area is :[0.94429,0.97872]
the8iteration search area is :[0.95744,0.97872]
the9iteration search area is :[0.96557,0.97872]
the10iteration search area is :[0.96557,0.9737]
the11iteration search area is :[0.96867,0.9737]
the12iteration search area is :[0.96867,0.97178]
the13iteration search area is :[0.96986,0.97178]
the14iteration search area is :[0.96986,0.97104]
the15iteration search area is :[0.97031,0.97104]
the16iteration search area is :[0.97031,0.97077]
the17iteration search area is :[0.97049,0.97077]
the18iteration search area is :[0.97059,0.97077]
the19iteration search area is :[0.97059,0.9707]
the20iteration search area is :[0.97063,0.9707]
the21iteration search area is :[0.97063,0.97067]
the22iteration search area is :[0.97065,0.97067]
the23iteration search area is :[0.97066,0.97067]
the24iteration search area is :[0.97066,0.97067]

x =
    0.9707
fval =
  -4.1086e+10
iter =
    24
```

即 $\min\limits_{x\in[0,1]} g(x)$值为$-4.1086e+10$。

【例 7-4】 求函数 $\varphi(t)=2e^{-t}+e^{t}$ 在$[-1,1]$内的极小值。

解: 编写函数代码如下:

```
function [dy,val] = funHJ(x)
val = exp(-x) + exp(x);
dy = -exp(-x) + exp(x);
```

根据函数和黄金分割算法代码,求解代码如下:

```
clear all
clc
[x,fval,iter] = HJ( - 1,1)
```

运行后得到结果如下：

```
the1iteration search area is :[ - 0.236,1]
the2iteration search area is :[ - 0.236,0.52785]
the3iteration search area is :[0.05579,0.52785]
the4iteration search area is :[0.23612,0.52785]
the5iteration search area is :[0.23612,0.41641]
the6iteration search area is :[0.30499,0.41641]
the7iteration search area is :[0.30499,0.37384]
the8iteration search area is :[0.33129,0.37384]
the9iteration search area is :[0.33129,0.35759]
the10iteration search area is :[0.34134,0.35759]
the11iteration search area is :[0.34134,0.35138]
the12iteration search area is :[0.34517,0.35138]
the13iteration search area is :[0.34517,0.34901]
the14iteration search area is :[0.34517,0.34754]
the15iteration search area is :[0.34608,0.34754]
the16iteration search area is :[0.34608,0.34698]
the17iteration search area is :[0.34642,0.34698]
the18iteration search area is :[0.34642,0.34677]
the19iteration search area is :[0.34642,0.34664]
the20iteration search area is :[0.34651,0.34664]
the21iteration search area is :[0.34656,0.34664]
the22iteration search area is :[0.34656,0.34661]
the23iteration search area is :[0.34656,0.34659]
the24iteration search area is :[0.34657,0.34659]
the25iteration search area is :[0.34657,0.34658]
the26iteration search area is :[0.34657,0.34658]

x =
    0.3466
fval =
    2.8284
iter =
    26
```

即函数 $\varphi(t)=2e^{-t}+e^{t}$ 在$[-1,1]$内极小值为0.3466。

【例 7-5】 函数 $f(x)=x^2-2x$，给定搜索区间为$[-2,6]$，精度为0.0001，求此函数的极小值点。

解：编写黄金分割算法代码如下：

```
function xmin = HJ(f,a,b,e)
k = 0;
a1 = b - 0.618 * (b - a);                    % 插入点的值
a2 = a + 0.618 * (b - a);
while b - a > e
    y1 = subs(f,a1);
    y2 = subs(f,a2);
    if y1 > y2                               % 比较插入点的函数值的大小
        a = a1;                              % 进行换名
        a1 = a2;
        y1 = y2;
        a2 = a + 0.618 * (b - a);
    else
        b = a2;
        a2 = a1;
        y2 = y1;
        a1 = b - 0.618 * (b - a);
    end
    k = k + 1;
end                                          % 迭代到满足条件为止就停止迭代
xmin = (a + b)/2;
fmin = subs(f,xmin)                          % 输出函数的最优值
fprintf('k = \n');
disp(k);
```

再编写主程序 MATLAB 代码如下：

```
syms x a b a3 e h;
a = input('搜索区间的第一点\a = ');
b = input('搜索区间的第二点\b = ');
e = input('搜索精度\ne = ');
f = input('函数\f = ');
disp('需求的优化函数 f = f(x),调用 xmin = HJ(f,a,b,e)');
```

MATLAB 运行得到结果如下：

```
>> ex7_5
搜索区间的第一点 a = -2
搜索区间的第二点 b = 6
搜索精度
e = 0.0001
函数 f = x^2 - 2 * x
需求的优化函数 f = f(x),调用 xmin = HJ(f,a,b,e)
>> xmin = HJ(f,a,b,e)

fmin =
 -1267650600052410009155277288327/1267650600228229401496703205376

k =
```

```
    24
xmin =
    1.0000
```

即该函数的极小值点在 $x=1$ 处，极小值为 -1，一共经历了 24 次迭代。

7.4 斐波那契法

斐波那契法(Fibonacci Method)又称斐波那契分数法，是一种一维搜索区间消去法。

斐波那契法通过取代试探点和进行函数值的比较，使包含极小值点的搜索区间不断缩短，当区间长度缩短到一定程度时，区间上各点的函数值均接近极小值点的近似。该算法要求所考虑的区间上的目标函数是单峰函数，即在这个区间上只有一个局部极小值点的函数。

斐波那契算法的具体步骤如下：

(1) 选取初始数据，确定单峰区间$[a_0,b_0]$，给出搜索精度 $\delta>0$，由步骤(4)确定搜索次数 n。

(2) $k=1,a=a_0,b=b_0$，计算最初两个搜索点，按步骤(3)计算 t_1 和 t_2。

(3) 计算程序如下：

```
while k<n-1
    f1 = f(t1),f2 = f(t2)
    if f1<f2
      a = t2;t2 = t1;t1 = a + F(n-1-k)/F(n-k)(b-a)
    else
      b = t1;t1 = t2;t2 = b + F(n-1-k)/F(n-k)(a-b)
end
k = k + 1
end
```

(4) 当进行至 $k=n-1$ 时，$t_1=t_2=\frac{1}{2}(a+b)$，这就无法借比较函数值 $f(t_1)$和 $f(t_2)$的大小来确定最终区间。为此，取

$$\begin{cases} t_2 = \frac{1}{2}(a+b) \\ t_1 = a + \left(\frac{1}{2}+\varepsilon\right)(b-a) \end{cases}$$

其中 ε 为任意小的数。在 t_1 和 t_2 这两点中，以函数值较小者为近似极小值点，相应的函数值为近似极小值。并得最终区间为$[a,t_1]$或$[t_2,b]$。

由上述分析可知，斐波那契法使用对称搜索的方法，逐步缩短所考察的区间，它能以尽量少的函数求值次数，达到预定的某一缩短率。

根据以上步骤，编写斐波那契算法 MATLAB 代码如下：

```
function[x,T,j] = Fibonacci(F_1,a1,b1,l,e)
n = 1;j = 1;
a(n) = a1;b(n) = b1;
```

```
while(Fib(j) * l<(b1 - a1))
    j = j + 1;
end
r(1) = a(1) + (1 - Fib(j - 1)/Fib(j)) * (b(1) - a(1));
u(1) = a(1) + Fib(j - 1)/Fib(j) * (b(1) - a(1));
for n = 1:1:j - 2
    R(n) = feval(F_1,r(n));
    U(n) = feval(F_1,u(n));
    Z(n) = b(n) - a(n);
if R(n)> U(n)
    a(n + 1) = r(n);
    b(n + 1) = b(n);
    r(n + 1) = u(n);
    u(n + 1) = a(n + 1) + Fib(j - n - 1)/Fib(j - n) * (b(n + 1) - a(n + 1));
else
    a(n + 1) = a(n);
    b(n + 1) = u(n);
    u(n + 1) = r(n);
    r(n + 1) = a(n + 1) + (1 - Fib(j - n - 1)/Fib(j - n)) * (b(n + 1) - a(n + 1));
end
end
R(j - 1) = feval(F_1,r(j - 1));
U(j - 1) = feval(F_1,u(j - 1));
r(j) = r(j - 1);
u(j) = r(j - 1) + e;
R(j) = feval(F_1,r(j));
U(j) = feval(F_1,u(j));
if R(j)> U(j)
    a(j) = r(j);
    b(j) = b(j - 1);
else
    a(j) = a(j - 1);
    b(j) = u(j);
end
Z(j - 1) = b(j - 1) - a(j - 1);
Z(j) = b(j) - a(j);
x = (a(j) + b(j))/2;
T = [a',b',r',u',R',U',Z'];

function Fi = Fib(n)
i = 1;
Fib(2) = 2;
Fib(1) = 1;
if n == 0
    Fi = 1;
else
    for i = 3:1:n
        Fib(i) = Fib(i - 1) + Fib(i - 2);
    end
```

```
    i = n;
end
Fi = Fib(i);
```

【例 7-6】 使用斐波那契法，求函数 $f(x)=x^2-4x+3$ 在区间[−2,1]上，精度不大于 0.15 的最小值点，要求最终区间长度不大于原始区间长度的 0.002 倍。

解：使用 MATLAB 编辑函数 $f(x)$代码如下：

```
function y = Fun(x)
y = x^2 - 4 * x + 3;
```

使用斐波那契法，在 MATLAB 中输入：

```
[x,T,j] = Fibonacci('Fun', - 2,1,0.15,0.02)
```

运行后得到结果如下：

```
x =
    0.9286
T =
   - 2.0000  1.0000  - 0.8571  - 0.1429  7.1633  3.5918  3.0000
   - 0.8571  1.0000  - 0.1429    0.2857  3.5918  1.9388  1.8571
   - 0.1429  1.0000    0.2857    0.5714  1.9388  1.0408  1.1429
     0.2857  1.0000    0.5714    0.7143  1.0408  0.6531  0.7143
     0.5714  1.0000    0.7143    0.8571  0.6531  0.3061  0.4286
     0.7143  1.0000    0.8571    0.8571  0.3061  0.3061  0.2857
     0.8571  1.0000    0.8571    0.8771  0.3061  0.2608  0.1429
j =
     7
```

即经过 7 次迭代，得到函数最小值为 0.9286。

7.5 牛顿型法

牛顿型法包括牛顿法和阻尼牛顿法。这类方法的最大优点是收敛速度快，即它的迭代次数相对于其他方法来说少得多。特别是对于一些性态较好的目标函数，例如二次函数，只需保证求梯度和二阶偏导数矩阵时的精度，不管初始点在何处，均可一步就找出最优点。可是这类方法也有很大的缺点，在每次迭代决定牛顿方向时，都要计算目标函数的一阶导数和二阶导数矩阵及其逆矩阵。这就使计算度较为复杂，增加了每次迭代的计算工作量和计算机存储量。

7.5.1 牛顿法

牛顿法是根据目标函数的等值线在极值点附近是同心椭圆族的特点，在极值点 X^*

邻域内用一个二次函数 $\varphi(X)$来近似代替原目标函数 $f(X)$，并将 $\varphi(X)$的极小值点作为对目标函数 $f(X)$求优的下一个迭代点，经多次迭代，使之逼近原目标函数 $f(X)$的极小值点。

牛顿法的基本原理如下：

设目标函数是连续二阶可微的，将函数在 $X^{(K)}$点按泰勒级数展开，并保留到二次项，得

$$f(X) \approx \varphi(X) = f(X^{(K)}) + [\nabla f(X^{(K)})]^{\mathrm{T}}(X - X^{(K)}) + \frac{1}{2}(X - X^{(K)})^{\mathrm{T}} \nabla^2 f(X^{(K)})(X - X^{(K)})$$

此式是个二次函数，设 $X^{(K+1)}$为 $\varphi(X)$的极小值点，则

$$\nabla\varphi(X^{(K+1)}) = 0$$

即

$$\nabla f(X^{(K)}) + \nabla^2 f(X^{(K)})(X^{(K+1)} - X^{(K)}) = 0$$

$$X^{(K+1)} = X^{(K)} - [\nabla^2 f(X^{(K)})]^{-1} \nabla f(X^{(K)}) \quad (K = 0,1,2,\cdots)$$

这就是多元函数求极值的牛顿法迭代公式。

式中取$\vec{d}^{(K)} = -[\nabla^2 f(X^{(K)})]^{-1}\nabla f(X^{(K)})$，称为牛顿方向，为常数。式中没有步长 α_K，或者可以看成步长恒等于1，所以牛顿法是一种定步长的迭代方法。

【例 7-7】 函数 $F(x)=\mathrm{e}^{(x_1^2+x_1+3x_2^2-3)}$，以 $\boldsymbol{x}_0=[1,2]^{\mathrm{T}}$ 为初始点，用牛顿法对其进行迭代。

解：根据题意编写 MATLAB 代码如下：

```
syms x1 x2
f = exp(x1 ^2 + x1 + 3 * x2 ^2 - 3);
v = [x1,x2];
df = jacobian(f,v);
df = df.';
G = jacobian(df,v);
epson = 1e - 12;
xm = [0,0]';
g1 = subs(df,{x1,x2},{xm(1,1),xm(2,1)});
G1 = subs(G,{x1,x2},{xm(1,1),xm(2,1)});
k = 0;
while(norm(g1)> epson)
    p = - G1\g1;
    xm = xm + p;
    g1 = subs(df,{x1,x2},{xm(1,1),xm(2,1)});
    G1 = subs(G,{x1,x2},{xm(1,1),xm(2,1)});
    k = k + 1;
end
k
xm
```

运行后得到结果如下：

```
k =
      4
xm =
   - 0.5
      0
```

即经过 4 次迭代，找到了该函数的极小值点。

7.5.2 阻尼牛顿法

对于二次函数，用牛顿法迭代一次即可得到最优点；对于非二次函数，若函数的迭代点已进入极小值点的邻域，则其收敛速度也是很快的。但是从牛顿法迭代公式的推导过程可以看出，迭代点是由近似二次函数 $\varphi(X)$ 的极值条件确定的，该点可能是 $\varphi(X)$ 极小值点，也可能是 $\varphi(X)$ 的极大值点。因此在用牛顿法迭代时，可能会出现函数上升的现象，即 $f(X^{(K+1)})>f(X^{(K)})$，使迭代不能收敛于最优点。

牛顿法不能保证函数值稳定地下降，在严重的情况下甚至不能收敛而导致计算失败。可见，牛顿法对初始点的要求是比较苛刻的，所选取的初始点离极小值点不能太远。而在极小值点位置未知的情况下，上述要求很难达到。

为了消除牛顿法的这些弊端，需要对其做一些修改。将牛顿法定步长的迭代，改为变步长的迭代，引入步长 α，在 $X^{(k)}$ 的牛顿方向进行一维搜索，保证每次迭代点的函数值都是下降的。这种方法称为阻尼牛顿法，其迭代公式为

$$X^{(K+1)} = X^{(K)} - \alpha_K[\nabla^2 f(X^{(K)})]^{-1}\nabla f(X^{(K)}) \quad (K = 0,1,2,\cdots)$$

式中，α_K 为牛顿方向的最优步长。这种方法对初始点的选取不再苛刻，从而提高了牛顿法的可靠度。但采用阻尼牛顿法，每次迭代都要进行一维搜索，使收敛速度大大降低。

例如，对于某些目标函数，取同样的初始点，采用阻尼牛顿法进行迭代，达到同样的精度，要经过多次的迭代，越靠近极小值点收敛速度越慢，使牛顿法收敛速度快的优势损失殆尽。

阻尼牛顿法的计算步骤如下：

(1) 给定初始点 $X^{(0)}$，收敛精度 ε，并令计算次数 $K=0$；

(2) 计算 $X^{(K)}$ 点的梯度 $\nabla f(X^{(K)})$ 和梯度的模 $\|\nabla f(X^{(K)})\|$；

(3) 判断是否满足精度指标 $\|\nabla f(X^{(K)})\|\leqslant\varepsilon$，若满足，$X^{(K)}$ 为最优点，迭代停止，输出最优解 $X^*=X^{(K)}$ 和 $f(X^*)=f(X^{(K)})$，否则进行下一步计算；

(4) 计算 $X^{(K)}$ 点的牛顿方向 $\vec{d}^{(K)}$，$\vec{d}^{(K)}=-[\nabla^2 f(X^{(K)})]^{-1}\nabla f(X^{(K)})$；

(5) 以 $X^{(K)}$ 为出发点，沿 $\vec{d}^{(K)}$ 进行一维搜索，求能使函数值下降最多的步长 α_K，即 $\min\limits_{\alpha} f(X^{(K)}+\alpha\vec{d}^{(K)})=f(X^{(K)}+\alpha_K\vec{d}^{(K)})$；

(6) 令 $X^{(K+1)}=X^{(K)}+\alpha_K\vec{d}^{(K)}$，$K=K+1$，转到步骤(2)。

根据以上步骤，阻尼牛顿法的 MATLAB 程序如下：

```
function [x,f,k] = dampnm(fun,gfun,Hess,x0)
maxk = 500;                    % 最大迭代次数
rho = 0.55;
sigma = 0.4;
k = 0;
epsilon = 1e-5;                % 计算精度
while(k < maxk)                % 判断迭代次数是否满足是定值
gk = feval(gfun,x0);           % 函数 f 在 x0 的梯度
Gk = feval(Hess,x0);           % 函数 f 在 x0 的海瑟矩阵(Hessian)
dk = - Gk\gk;                  % 解方程组 - gk = Gk * dk
if(norm(gk)< epsilon),break; % 判断终止迭代准则,是否满足设定精度
end;
m = 0;mk = 0;
while(m < 20)                  % 运用 Armijo 法做非精度线搜索,确定步长因子
if(feval(fun,x0 + rho ^ m * dk)< feval(fun,x0) + sigma * rho ^ m * gk' * dk)
mk = m;
break;
end
m = m + 1;
end
x0 = x0 + rho ^ mk * dk;
k = k + 1;
end
x = x0;                        % 赋值最后迭代点
f = feval(fun,x);              % 计算最后迭代点 x 的函数值
```

【例 7-8】 已知无约束优化问题的目标函数是 $f(x)=50(x_1^2-x_2)^2+(x_1-3)^2$,求在初始迭代点 $\boldsymbol{x}^{(0)}=[1,1]^{\mathrm{T}}$ 下,迭代次数不超过 500 次的目标函数最优值。

解:在 MATLAB 中依次建立目标函数代码

建立目标函数 $f(x)$的 MATLAB 代码:

```
function f = fun(x)
f = 50 * (x(1)^2 - x(2))^2 + (x(1) - 3)^2;
```

建立目标函数梯度的 MATLAB 代码:

```
function g = mfun(x)
g = [400 * x(1) * (x(1)^2 - x(2)) + 2 * (x(1) - 1), - 200 * (x(1)^2 - x(2))]';
```

建立目标函数 Hessian 矩阵的 MATLAB 代码:

```
function He = Hfun(x)
n = length(x);
He = zeros(n,n);
He = [1200 * x(1)^2 - 400 * x(2) + 2, - 400 * x(1); - 400 * x(1),200];
```

调用上面的程序,在 MATLAB 命令窗口中输入:

```
clear all
clc
x0 = [1 1]';
[x,f,k] = dampnm('fun','mfun','Hfun',x0)
```

运行后得到结果如下：

```
x =
    1.0000
    1.0000
f =
    4.0000
k =
    25
```

得到在 $\boldsymbol{x}^{(0)}=[1,1]^{\mathrm{T}}$ 下，迭代次数不超过 500 次的目标函数最优值为 4。

使用 MATLAB 中提供的优化工具箱 fminunc 函数，也可以实现上述结果。

调用 fminunc 函数程序代码如下：

```
clear all
clc
x0 = [0 1];
[x,fval] = fminunc(@fun,x0)
```

运行后得到结果如下：

```
x =
    2.9999   8.9996
fval =
   3.6139e-09
```

7.6 割线法

上一节讲了牛顿型法拥有许多良好的性质，如二次收敛速度很快、迭代次数较少、所用存储空间少、程序编写简单等。但不可否认的是，牛顿法仍然存在一些缺点，如局部收敛，需要求函数零点导数等。因此为克服当函数不可导时无法应用牛顿法这一缺点，人们提出了割线法，即

$$x_{n-1}=x_n-f(x_n)\frac{x_n-x_{n-1}}{f(x_n)-f(x_{n-1})}$$

割线法中 x_{n-1} 计算需要用到 x_n 和 x_{n-1}，即需要制定两个初始点。

编写割线法的 MATLAB 代码如下：

```
%f 是给定的非线性函数
function[p1,err,k,y] = secant(f,p0,p1,delta,max1)
```

```
%p0,p1为初始值
%delta为给定误差界
%max1迭代次数的上限
%p1为所求得的方程的近似解
%err为p1-p0的绝对值
%k为所需要的迭代次数
%y=f(p1)
k=0,
p0,
p1,
feval('f',p0),
for k=1:max1
    p2=p1-feval('f',p1)*(p1-p0)/(feval('f',p1)-feval('f',p0))
    err=abs(p2-p1);
    p0=p1;
    p1=p2;
k,
p1,
err,
y=feval('f',p1)
if(err<delta|(y==0))
    break,
end
end
```

【例 7-9】 使用割线法求解非线性方程 $f(x)=x^3+x-2=0$，给定初始位置为 p0=1.5，p1=−1.53，误差设定为 10^{-7}。

解：首先编写非线性方程 MATLAB 代码如下：

```
function y=f(x)
y=x^3+x-2;
```

然后在 MATLAB 命令窗口中输入：

```
secant('f',-1.5,-1.53,10^(-7),3)
```

运行后得到结果如下：

```
k =
     0
p0 =
   -1.5000
p1 =
   -1.5300
ans =
   -6.8750
p2 =
   -0.6282
k =
```

```
        1
    p1 =
      - 0.6282
    err =
        0.9018
    y =
      - 2.8761
    p2 =
      - 0.0158
    k =
        2
    p1 =
      - 0.0158
    err =
        0.6124
    y =
      - 2.0158
    p2 =
        1.4191
    k =
        3
    p1 =
        1.4191
    err =
        1.4349
    y =
        2.2771
    ans =
        1.4191
```

以上结果表明，经过 3 次迭代得到了满足精度要求的近似解 $x^* \approx x_3 = 1.4191$，且 $f(x_3) = 2.2771$。

7.7 抛物线法

抛物线法是求无约束一维极值问题的一种方法，也叫二次插值法，其理论依据为二次多项式可以在最优点附近较好地逼近函数的形状，做法是在函数的最优点附近取三个构造点，然后用这三个点构造一条抛物线，把这条抛物线的极值点作为函数的极值点的近似。

每次构造一条抛物线后，抛物线的极值点就可作为一个新的构造点，新的构造点与原来的三个构造点经过某种算法，得到下一步抛物线逼近的三个构造点，这就是抛物线法的算法过程。

编写抛物线法的 MATLAB 代码如下：

```
function root = Parabola(f,a,b,x,eps)
% 抛物线法求函数 f 在区间[a,b]上的一个零点
```

```
%函数名: f
%区间左端点: a
%区间右端点: b
%初始迭代点: x
%根的精度: eps
%求出的函数零点: root
if(nargin == 4)
eps = 1.0e-4;
end
f1 = subs(sym(f),findsym(sym(f)),a);
f2 = subs(sym(f),findsym(sym(f)),b);
if(f1 == 0)
root = a;
end
if(f2 == 0)
root = b;
end
if(f1 * f2 > 0)
disp('两端点函数值乘积大于0');
return;
else
tol = 1;
fa = subs(sym(f),findsym(sym(f)),a);
fb = subs(sym(f),findsym(sym(f)),a);
fx = subs(sym(f),findsym(sym(f)),x);
d1 = (fb - fa)/(b - a);
d2 = (fx - fb)/(x - b);
d3 = (f2 - f1)/(x - a);
B = d2 + d3 * (x - b);
root = x - 2 * fx/(B + sign(B) * sqrt(B^2 - 4 * fx * d3));
t = zeros(3);
t(1) = a;
t(2) = b;
t(3) = x;
while(tol > eps)
t(1) = t(2);                                    %保存3个点
t(2) = t(3);
t(3) = root;
f1 = subs(sym(f),findsym(sym(f)),t(1));         %计算3个点的函数值
f2 = subs(sym(f),findsym(sym(f)),t(2));
f3 = subs(sym(f),findsym(sym(f)),t(3));
d1 = (f2 - f1)/(t(2) - t(1));                   %计算3个差分
d2 = (f3 - f2)/(t(3) - t(2));
d3 = (d2 - d1)/(t(3) - t(1));
B = d2 + d3 * (t(3) - t(2));                    %计算算法中的B
root = t(3) - 2 * f3/(B + sign(B) * sqrt(B^2 - 4 * f3 * d3));
tol = abs(root - t(3));
end
end
```

【例 7-10】 采用抛物线法求方程 $\lg x+\sqrt{x}=2$ 在区间[1,4]上的一个根。

解：根据抛物线算法，编写代码如下：

```
clear all
clc
r = Parabola('sqrt(x) + log(x) - 1',1,3,2)
```

运行后得到结果如下：

```
r =
    1.0000
```

即方程在区间[1,4]上的一个根为 1。

7.8 三次插值法

在许多问题中，通常根据实验、观测或经验得到的函数表或离散点上的信息，去研究分析函数的有关特性。其中插值法是一种最基本的方法，以下给出最基本的插值问题——三次样条插值的基本方法：

对插值区间$[a,b]$进行划分，$a\leqslant x_0<x_1<\cdots<x_n\leqslant b$，函数 $y=f(x)$在节点 x_i 上的值 $y_i=f(x_i)(i=0,1,2,\cdots,n)$，并且如果函数 $S(x)$在每个小区间$[x_i,x_{i+1}]$上是三次多项式，在$[a,b]$上有二阶连续导数，则称 $S(x)$是$[a,b]$上的三次样条函数，如果 $S(x)$在节点 x_i 上还满足条件 $S(x_i)=y_i(i=0,1,\cdots,n)$，则称 $S(x)$为三次样条插值函数。

【例 7-11】 已知一组数据点(x_i,y_i)，$i=1,\cdots,n$，编写一个程序求解三次样条插值函数 $S(x)$，且满足 $S(x_i)=y_i(i=1,\cdots,n)$并针对下面一组具体实验数据求解，其中边界条件为 $S''(0.25)=0$，$S''(0.53)=0$。

x_i	0.25	0.3	0.39	0.45	0.53
y_i	0.5000	0.5477	0.6245	0.6708	0.7280

解：根据题意，编写 MATLAB 代码如下：

```
clear all
clc
x = [0.25 0.3 0.39 0.45 0.53]
y = [0.5000 0.5477 0.6245 0.6708 0.7280]
n = length(x);
for i = 1:n - 1
    h(i) = x(i + 1) - x(i);
end
for i = 1:n - 2
    k(i) = h(i + 1)/(h(i) + h(i + 1));
    u(i) = h(i)/(h(i) + h(i + 1));
```

```
end
for i = 1:n - 2
    g1(i) = 3 * (u(i) * (y(i + 2) - y(i + 1))/h(i + 1) + k(i) * (y(i + 1) - y(i))/h(i));
end
g0 = 3 * (y(2) - y(1))/h(1);
g00 = 3 * (y(n) - y(n - 1))/h(n - 1);
g = [g0 g1 g00];
g = transpose(g)
k1 = [k 1];
u1 = [1 u];
Q = 2 * eye(5) + diag(u1,1) + diag(k1, - 1)
m = transpose(Q\g)
syms X;
for i = 1:n - 1
p1(i) = (1 + 2 * (X - x(i))/h(i)) * ((X - x(i + 1))/h(i))^2 * y(i);
p2(i) = (1 - 2 * (X - x(i + 1))/h(i)) * ((X - x(i))/h(i))^2 * y(i + 1);
p3(i) = (X - x(i)) * ((X - x(i + 1))/h(i))^2 * m(i);
p4(i) = (X - x(i + 1)) * ((X - x(i))/h(i))^2 * m(i + 1);
p(i) = p1(i) + p2(i) + p3(i) + p4(i);
end
s1 = p(1)
s2 = p(2)
s3 = p(3)
s4 = p(4)
for k = 1:4
    for z = x(k):0.001:x(k + 1)
        q = eval(subs(p(k),'X','z'));
    end
end
```

运行后得到结果如下：

```
x =
    0.2500   0.3000   0.3900   0.4500   0.5300
y =
    0.5000   0.5477   0.6245   0.6708   0.7280
g =

    2.8620
    2.7541
    2.4130
    2.2421
    2.1450
Q =
    2.0000   1.0000        0        0        0
    0.6429   2.0000   0.3571        0        0
         0   0.4000   2.0000   0.6000        0
         0        0   0.5714   2.0000   0.4286
         0        0        0   1.0000   2.0000
```

```
m =
    0.9697   0.9227   0.7992   0.7424   0.7013
s1 =
(8733947062530765 * (20 * X - 6)^2 * (X - 1/4))/9007199254740992 + (4155355071003587 *
(20 * X - 5)^2 * (X - 3/10))/4503599627370496 + ((20 * X - 6)^2 * (40 * X - 9))/2 -
(5477 * (20 * X - 5)^2 * (40 * X - 13))/10000
s2 =
(4155355071003587 * ((100 * X)/9 - 13/3) ^ 2 * (X - 3/10))/4503599627370496 +
(7198836265065077 * ((100 * X)/9 - 10/3)^2 * (X - 39/100))/9007199254740992 + (5477 *
((100 * X)/9 - 13/3)^2 * ((200 * X)/9 - 17/3))/10000 - (1249 * ((100 * X)/9 - 10/3)^2 *
((200 * X)/9 - 29/3))/2000

s3 =
(6687358691261619 * ((50 * X)/3 - 13/2) ^ 2 * (X - 9/20))/9007199254740992 +
(7198836265065077 * ((50 * X)/3 - 15/2)^2 * (X - 39/100))/9007199254740992 + (1249 *
((50 * X)/3 - 15/2)^2 * ((100 * X)/3 - 12))/2000 - (1677 * ((50 * X)/3 - 13/2)^2 * ((100
 * X)/3 - 16))/2500
s4 =
(6687358691261619 * ((25 * X)/2 - 53/8) ^ 2 * (X - 9/20))/9007199254740992 +
(6316541855078907 * ((25 * X)/2 - 45/8)^2 * (X - 53/100))/9007199254740992 + (1677 *
(25 * X - 41/4) * ((25 * X)/2 - 53/8)^2)/2500 - (91 * ((25 * X)/2 - 45/8)^2 * (25 * X -
57/4))/125
```

7.9 坐标轮换法

坐标轮换法是将多维问题转化为一系列一维问题的求解方法，它将多变量的优化问题轮流转化为单变量的优化问题，因此又称为变量轮换法。这种方法在搜索过程中只需要目标函数的信息，而不需要求解目标函数的导数。

坐标轮换法轮流沿坐标方向搜索，每次只允许一个变量变化，其余变量保持不变。以二元函数 $f(x_1,x_2)$ 为例，说明坐标轮换法的迭代过程。

如图 7-1 所示，选定的初始点 $X^{(0)}$ 作为第一轮的始点 $X_0^{(1)}$，保持 X_2 不变而沿 X_1 方向 $\boldsymbol{e}_1=[1,0]^{\mathrm{T}}$ 作一维搜索，确定其最优步长 $\alpha_1^{(1)}$，即可获得第一轮的第一个迭代点 $X_1^{(1)}=X_0^{(1)}+\alpha_1^{(1)}\boldsymbol{e}_1$。

然后以 $X_1^{(1)}$ 为新起点，改沿 X_2 方向 $\boldsymbol{e}_1=[0,1]^{\mathrm{T}}$ 作一维搜索，确定其最优步长 $\alpha_2^{(1)}$，可得第一轮的第二个迭代点 $X_2^{(1)}=X_1^{(1)}+\alpha_2^{(1)}\boldsymbol{e}_2$。这个二维问题经过沿和 $\boldsymbol{e}_1$ 和 $\boldsymbol{e}_2$ 方向的两次一维搜索完成了第一轮迭代。接着的第二轮迭代则是 $X_0^{(2)}\Leftarrow X_2^{(1)}$，$X_1^{(2)}=X_0^{(2)}+\alpha_1^{(2)}\boldsymbol{e}_1$，$X_2^{(2)}=X_1^{(2)}+\alpha_2^{(2)}\boldsymbol{e}_2$。

按照同样的方式进行第三轮、第四轮……迭代。随着迭代的进行，目标函数值不断下降，最后的迭代点必将逼近该二维目标函数的最优点。

迭代的终止准则可以采用点距准则，即在一轮迭代后，终点与始点的距离小于收敛精度 ε，则迭代停止。

对 n 维优化问题，先将$(n-1)$个变量固定不动，只对第一个变量进行一维搜索得到

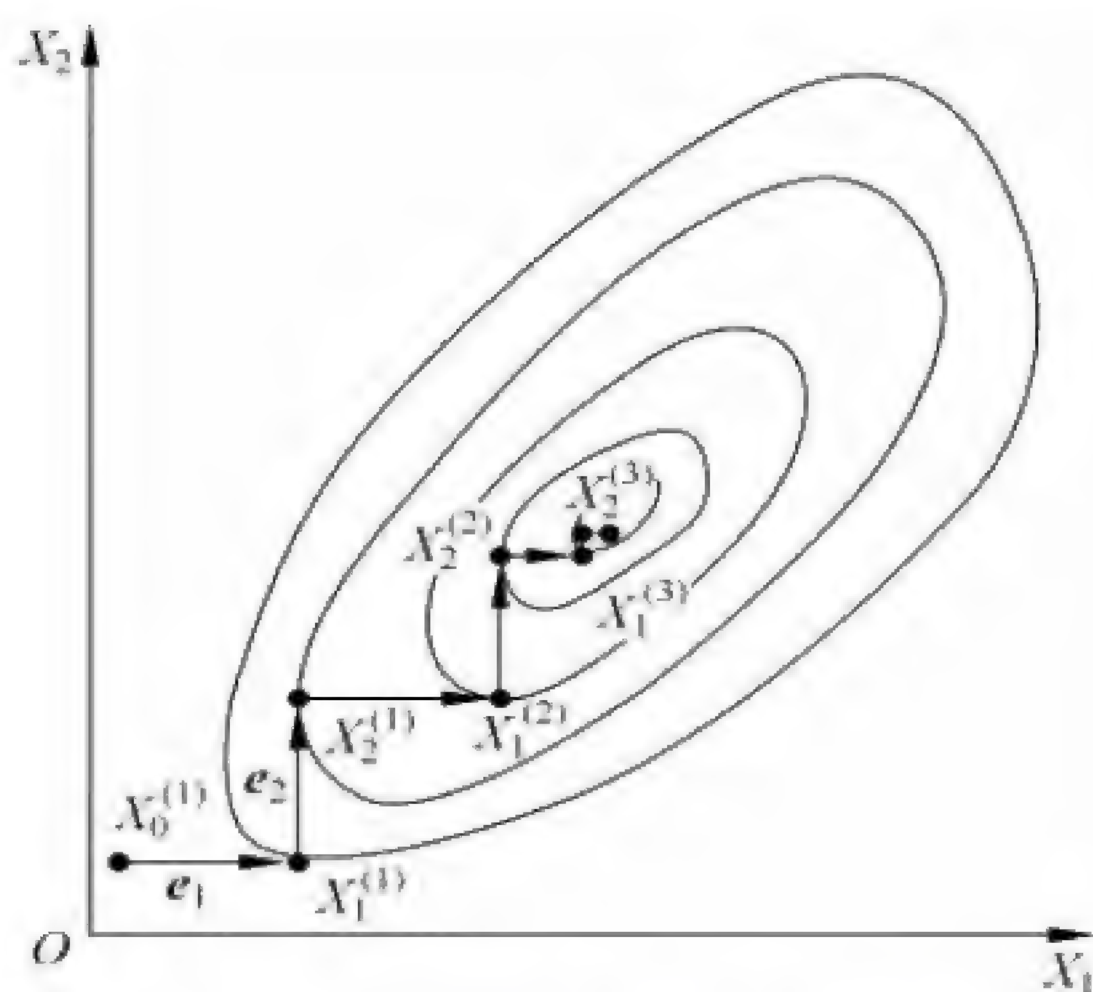


图 7-1　坐标轮换法的搜索过程

最优点 $X_1^{(1)}$。然后，再对第二个变量进行一维搜索到 $X_2^{(1)}$ 点等。总之，每次都固定 $(n-1)$ 个变量不变，只对目标函数的一个变量进行一维搜索，当 n 个变量 $X_1, X_2, \cdots, X_n$ 依次进行过一轮搜索之后，即完成一轮迭代计算。若未收敛，则又从前一轮的最末点开始，做下一轮迭代计算，如此继续下去，直至收敛到最优点为止。

坐标轮换法的迭代过程如下：

(1) 取初始点 $X^{(0)}$，计算精度 ε。取搜索方向为 n 个坐标的单位向量，$\vec{d}_i = \boldsymbol{e}_i = [0, \cdots, 1, \cdots, 0]^{\mathrm{T}}$。

(2) 按公式 $X_i^{(k)} = X_{i-1}^{(k)} + \alpha_i^{(k)} \vec{d}_i^{(k)} (i = 0, 1, \cdots, n)$ 进行迭代计算。其中 $\alpha_i^{(k)}$ 可通过一维搜索的方法确定。

(3) 判断是否满足 $\| X_n^{(k)} - X_0^{(k)} \| \leqslant \varepsilon$。若满足，迭代终止，输出最优解 $X^* = X_n^{(k)}$；否则 $k = k + 1$，返回步骤(2)。

坐标轮换法具有算法简单、易于实现的优点。但是采用坐标轮换法，只能轮流沿坐标方向搜索，尽管它具有步步下降的特点，但路程迂回曲折，要多次变换方向，才有可能求得无约束极值点，尤其在极值点附近，每次搜索的步长更小，因此收敛很慢。

此外，坐标轮换法的收敛效率在很大程度上取决于目标函数的形态。若目标函数的等值线为长短轴都平行于坐标轴的椭圆形，则这种搜索方法很有效，两次就可达到极值点。但当目标函数的等值线近似于椭圆且长短轴倾斜时，用这种搜索方法，必须多次迭代才能曲折地达到最优点。

当目标函数的等值线出现脊线时，这种搜索方法完全无效。因为每次的搜索方向总是平行于某一坐标轴，不会斜向前进，所以一旦遇到了等值线的脊线，就不能找到更好的点。

【例 7-12】 使用坐标轮换法，求解下面目标函数的无约束最优解。其中，设定精度为 0.0001，初始点为[4,2]。

$$f(x) = 10x_1^2 + 106x_2^2 + 10x_1x_2 + 96x_1 + 100x_2$$

解：根据题意输入代码如下：

```
clear all
clc
e = input('输入精度要求 e: ');
X = input('输入初始点: ');
syms t s
a = 10 * X(1,1)^2 + 106 * X(2,1)^2 + 10 * X(1,1) * X(2,1) + 96 * X(1,1) + 100 * X(2,1);
k = 1;
e1 = [1;0];
e2 = [0;1];
A = X;                    % A 矩阵用于存储每一轮变换所得解
C = X + t * e1;           % 沿 e1 方向搜索
x1 = C(1,1);
x2 = C(2,1);
df = diff(10 * x1 ^2 + 106 * x2 ^2 + 10 * x1 * x2 + 96 * x1 + 100 * x2);
t = solve(df);
X = X + t * e1;
C = X + s * e2;           % 沿 e2 方向搜索
x1 = C(1,1);
x2 = C(2,1);
df = diff(10 * x1 ^2 + 106 * x2 ^2 + 10 * x1 * x2 + 96 * x1 + 100 * x2);
s = solve(df);
X = X + s * e2;
A = [A X];
b = 10 * X(1,1)^2 + 106 * X(2,1)^2 + 10 * X(1,1) * X(2,1) + 96 * X(1,1) + 100 * X(2,1);
a = [a b];
B = A(:,k + 1) - A(:,k);
while double(sqrt(B(1,1)^2 + B(2,1)^2))> e
syms t s
C = X + t * e1;           % 沿 e1 方向搜索
x1 = C(1,1);
x2 = C(2,1);
df = diff(10 * x1 ^2 + 106 * x2 ^2 + 10 * x1 * x2 + 96 * x1 + 100 * x2);
t = solve(df);
X = X + t * e1;
C = X + s * e2;           % 沿 e2 方向搜索
x1 = C(1,1);
x2 = C(2,1);
df = diff(10 * x1 ^2 + 106 * x2 ^2 + 10 * x1 * x2 + 96 * x1 + 100 * x2);
s = solve(df);
X = X + s * e2;
A = [A X];
b = 10 * X(1,1)^2 + 106 * X(2,1)^2 + 10 * X(1,1) * X(2,1) + 96 * X(1,1) + 100 * X(2,1);
a = [a b];
B = A(:,k + 1) - A(:,k);
k = k + 1;
end
```

```
Y=10*X(1,1)^2+106*X(2,1)^2+10*X(1,1)*X(2,1)+96*X(1,1)+100*X(2,1);
A
a
fprintf('轮换次数k=%f\n',k);
X
Y
```

运行后得到结果如下：

```
A =
[ 4, - 29/5,  - 4983/1060,  - 1050571/224720, - 222691927/47640640,  - 47210542899/
10099815680, -10008634366463/2141160924160]
[ 2, -21/106, -5617/22472, -1196629/4764064, -253714473/1009981568, -53787613901/
214116092416, -11402974875137/45392611592192]

a =
[ 1248, -119017/530, -5643617873/23820320, -253654124629737/1070580462080,
-      114002311614325403353/48116168287723520,       -      51237198932402593464O857/
216253306752344588288O, -2302804668817913665252406763 3/971928861867737517601587 20]

轮换次数k=6.000000
X =
   -4.6744
   -0.2512
Y =

   -236.9314
```

本章小结

在最优化计算中一维最优化方法是优化设计中最简单、最基本的方法。一维极值，又称为线性搜索。一维问题是多维问题的基础，在数值方法迭代计算过程中，都要进行一维极值优化，也可以把多维问题化为一维问题来处理。一维问题算法的好坏，直接影响到最优化问题的求解速度。

约束优化问题的求解可以通过一系列无约束优化方法来达到，所以无约束优化问题的解法是优化设计方法的基本组成部分，也是优化方法的基础。

第8章 无约束多维极值

在数学优化问题中，有些实际问题，其数学模型本身就是一个无约束优化问题。通过熟悉它的解法可以为研究约束优化问题打下良好的基础。

本章主要讲解了三种不同的无约束多维极值算法，主要包括直接法、间接法和拟牛顿法。

学习目标：

(1) 了解和掌握直接法的应用；

(2) 了解和掌握间接法的应用；

(3) 了解和掌握拟牛顿的应用。

8.1 直接法

对于无约束最优化问题，经常会使用一种方法——直接法，即不用计算导数，只需要计算函数值的方法。本节将重点讲述几种主要的直接法。

8.1.1 模式搜索法

模式搜索法是 Hooks 和 Jeeves 于 1961 年提出来的，模式搜索法每一次迭代都是交替进行轴向移动和模式移动。轴向移动的目的是探测有利的下降方向，而模式移动的目的则是沿着有利方向加速移动。在几何上是寻找具有较小函数值的“山谷”，力图使迭代产生的序列沿“山谷”逼近极小值点。

这种方法类似最速下降法，沿瘠线下山是步长加速的有利方向。对峰回路转，“褶皱”特多的曲面而言，这为跳出某一局部山谷而达到另一可能更优山谷提供了可能。

虽然模式搜索法可能要计算很多点的函数值才能得到目标函数的近似极小值点。但是它易于在计算机上实现，实际效果也不错，因此被认为是一种可靠的方法。

运用 MATLAB 工具箱里的 patternsearch，可实现模式搜索法。

patternsearch 函数的完整格式为

```
x = patternsearch(fun,x0)
```

其中,fun 为目标函数,x0 为起始点。

【例 8-1】 使用模式搜索法求解函数 $e^{(-x_1^2-x_2^2)(1+5x_1+6x_2+12x_1\cos x_2)}$。

解:在 MATLAB 中编写代码如下:

```
function y = psobj(x)
y = exp( - x(1)^2 - x(2)^2) * (1 + 5 * x(1)  + 6 * x(2)  + 12 * x(1) * cos(x(2)));

clear all
clc
fun = @psobj;                  % 建立目标函数
x0 = [0,0];                    % 起始点
x = patternsearch(fun,x0)      % 求解
```

运行后得到结果如下:

```
Optimization terminated: mesh size less than options.MeshTolerance.
x =
   - 0.7037   - 0.1860
```

即函数最优解为—0.7037 和—0.186。

【例 8-2】 使用模式搜索法求解函数 $e^{(-x_1^2-2x_2^2)(1-5x_1-3x_2+9x_1\cos x_2)}$。

解:运用函数 patternsearch,在 MATLAB 中编写代码如下:

```
% 目标函数
function y = psobj(x)
y = exp( - x(1)^2 - 2 * x(2)^2) * (1 - 5 * x(1)  - 3 * x(2)  + 9 * x(1) * cos(x(2)));

% 主函数
clear all
clc
fun = @psobj;
options = optimoptions('patternsearch','Display','iter','PlotFcn',@psplotbestf);
x0 = [0,0];
A = [];
b = [];
Aeq = [];
beq = [];
lb = [];
ub = [];
nonlcon = [];
x = patternsearch(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

运行后得到结果如下:

```
Iter     f-count      f(x)      MeshSize     Method
   0        1            1           1
   1        3    -0.270671           2      Successful Poll
   2        7    -0.270671           1      Refine Mesh
   3       11    -0.270671         0.5      Refine Mesh
   4       15    -0.303265           1      Successful Poll
   5       18    -0.758251           2      Successful Poll
   6       22    -0.758251           1      Refine Mesh
   7       26    -0.758251         0.5      Refine Mesh
   8       27      -0.9207           1      Successful Poll
   9       31      -0.9207         0.5      Refine Mesh
  10       35      -0.9207        0.25      Refine Mesh
  11       38       -0.924         0.5      Successful Poll
  12       42     -1.13957           1      Successful Poll
  13       46     -1.13957         0.5      Refine Mesh
  14       50     -1.13957        0.25      Refine Mesh
  15       52     -1.27727         0.5      Successful Poll
  16       56     -1.27727        0.25      Refine Mesh
  17       60     -1.27727       0.125      Refine Mesh
  18       64     -1.28251        0.25      Successful Poll
  19       68     -1.28251       0.125      Refine Mesh
  20       72     -1.28251      0.0625      Refine Mesh
  21       74      -1.2981       0.125      Successful Poll
  22       78      -1.2981      0.0625      Refine Mesh
  23       82      -1.2981     0.03125      Refine Mesh
  24       85     -1.29824      0.0625      Successful Poll
  25       89     -1.29824     0.03125      Refine Mesh
  26       93     -1.29824     0.01563      Refine Mesh
  27       94     -1.29886     0.03125      Successful Poll
  28       98     -1.29886     0.01563      Refine Mesh
  29      102     -1.29904     0.03125      Successful Poll
  30      106     -1.29904     0.01563      Refine Mesh

Iter     f-count      f(x)      MeshSize     Method
  31      110     -1.29904    0.007813      Refine Mesh
  32      112     -1.29923     0.01563      Successful Poll
  33      116     -1.29923    0.007813      Refine Mesh
  34      120     -1.29923    0.003906      Refine Mesh
  35      123     -1.29928    0.007813      Successful Poll
  36      127     -1.29928    0.003906      Refine Mesh
  37      131     -1.29928    0.007813      Successful Poll
  38      135     -1.29928    0.003906      Refine Mesh
  39      139     -1.29928    0.001953      Refine Mesh
  40      141      -1.2993    0.003906      Successful Poll
  41      145      -1.2993    0.001953      Refine Mesh
  42      149      -1.2993   0.0009766      Refine Mesh
  43      152      -1.2993    0.001953      Successful Poll
  44      156      -1.2993   0.0009766      Refine Mesh
  45      160      -1.2993   0.0004883      Refine Mesh
  46      161      -1.2993   0.0009766      Successful Poll
```

```
   47       165           - 1.2993     0.0004883      Refine Mesh
   48       169           - 1.2993     0.0009766      Successful Poll
   49       173           - 1.2993     0.0004883      Refine Mesh
   50       176           - 1.2993     0.0009766      Successful Poll
   51       180           - 1.2993     0.0004883      Refine Mesh
   52       184           - 1.2993     0.0002441      Refine Mesh
   53       185           - 1.2993     0.0004883      Successful Poll
   54       189           - 1.2993     0.0002441      Refine Mesh
   55       191           - 1.2993     0.0004883      Successful Poll
   56       195           - 1.2993     0.0002441      Refine Mesh
   57       199           - 1.2993     0.0001221      Refine Mesh
   58       203           - 1.2993     0.0002441      Successful Poll
   59       207           - 1.2993     0.0001221      Refine Mesh
   60       210           - 1.2993     0.0002441      Successful Poll

Iter      f - count        f(x)        MeshSize       Method
   61       214           - 1.2993     0.0001221      Refine Mesh
   62       218           - 1.2993     6.104e - 05     Refine Mesh
   63       219           - 1.2993     0.0001221      Successful Poll
   64       223           - 1.2993     6.104e - 05     Refine Mesh
   65       227           - 1.2993     3.052e - 05     Refine Mesh
   66       231           - 1.2993     1.526e - 05     Refine Mesh
   67       235           - 1.2993     3.052e - 05     Successful Poll
   68       239           - 1.2993     1.526e - 05     Refine Mesh
   69       242           - 1.2993     3.052e - 05     Successful Poll
   70       246           - 1.2993     1.526e - 05     Refine Mesh
   71       250           - 1.2993     7.629e - 06     Refine Mesh
   72       251           - 1.2993     1.526e - 05     Successful Poll
   73       255           - 1.2993     7.629e - 06     Refine Mesh
   74       259           - 1.2993     3.815e - 06     Refine Mesh
   75       263           - 1.2993     1.907e - 06     Refine Mesh
   76       265           - 1.2993     3.815e - 06     Successful Poll
   77       269           - 1.2993     1.907e - 06     Refine Mesh
   78       272           - 1.2993     3.815e - 06     Successful Poll
   79       276           - 1.2993     1.907e - 06     Refine Mesh
   80       280           - 1.2993     9.537e - 07     Refine Mesh
Optimization terminated: mesh size less than options.MeshTolerance.

x =

   - 0.7703    0.1774
```

最优函数值变化如图 8-1 所示。

8.1.2 单纯形搜索法

单纯形方法的基本思想是从一个基本可行解出发，求一个使目标函数值有所改善的基本可行解。通过不断改进基本可行解，力图达到最优基本可行解。

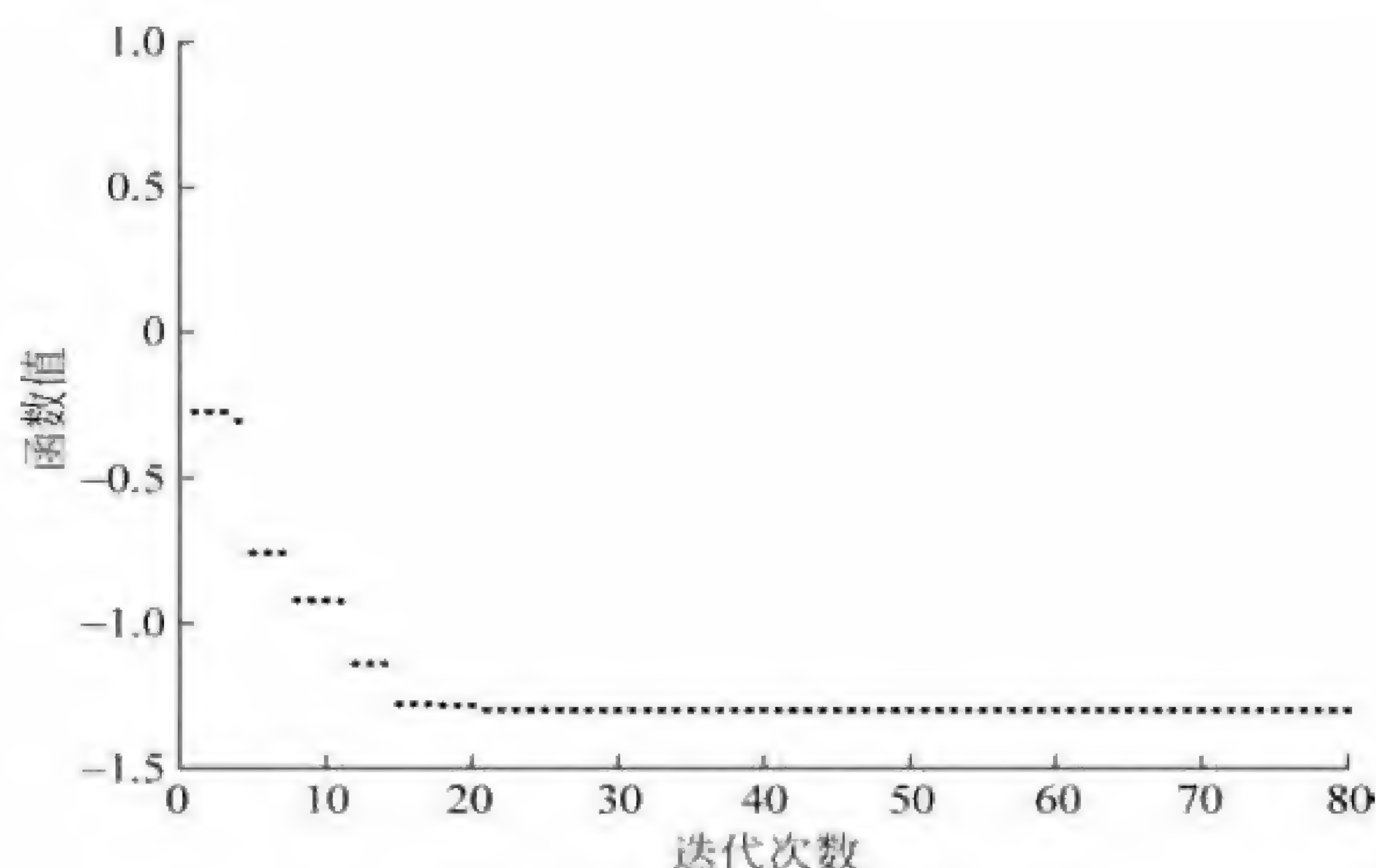


图 8-1　最优函数值变化图

对于问题

$$\min f \overset{\text{def}}{=\!=} \boldsymbol{cx}$$

$$\text{s. t.} \begin{cases} \boldsymbol{Ax} = \boldsymbol{b} \\ x \geqslant 0 \end{cases}$$

$\boldsymbol{A}$ 是一个 $m\times n$ 矩阵，且秩为 m，$\boldsymbol{c}$ 为 n 维行向量，$\boldsymbol{x}$ 为 n 维列向量，$\boldsymbol{b}$ 为 m 维非负列向量。符号$\overset{\text{def}}{=\!=}$表示右端的表达式是左端的定义式，即目标函数 f 的具体形式就是 $\boldsymbol{cx}$。

设定 $\boldsymbol{A}=(p_1,p_2,\cdots,p_n)$，令 $\boldsymbol{A}=(\boldsymbol{B},\boldsymbol{N})$，$\boldsymbol{B}$ 为基矩阵，$\boldsymbol{N}$ 为非基矩阵，设 $\boldsymbol{x}^{(0)}=\begin{bmatrix}\boldsymbol{B}^{-1}\boldsymbol{b}\\0\end{bmatrix}$是基本可行解，在 $\boldsymbol{x}^{(0)}$ 处的目标函数值

$$f_0=\boldsymbol{cx}^{(0)}=(\boldsymbol{c_B},\boldsymbol{c_N})\begin{bmatrix}\boldsymbol{B}^{-1}\boldsymbol{b}\\0\end{bmatrix}=\boldsymbol{c_B}\boldsymbol{B}^{-1}\boldsymbol{b}。$$

其中 $\boldsymbol{c_B}$ 是 $\boldsymbol{c}$ 中与基变量对应的分量组成的 m 维行向量；$\boldsymbol{c_N}$ 是 $\boldsymbol{c}$ 中与非基变量对应的分量组成的 $n-m$ 维行向量。

现由基本可行解 $\boldsymbol{x}^{(0)}$ 出发求解一个改进的基本可行解。

设 $\boldsymbol{x}=\begin{bmatrix}\boldsymbol{x_B}\\\boldsymbol{x_N}\end{bmatrix}$是任一可行解，则由 $\boldsymbol{Ax}=\boldsymbol{b}$ 得到 $\boldsymbol{x_B}=\boldsymbol{B}^{-1}\boldsymbol{b}-\boldsymbol{B}^{-1}\boldsymbol{N}\boldsymbol{x_N}$，在点 $\boldsymbol{x}$ 处的目标函数值 $f=\boldsymbol{cx}=(\boldsymbol{c_B},\boldsymbol{c_N})\begin{bmatrix}\boldsymbol{x_B}\\\boldsymbol{x_N}\end{bmatrix}=f_0-\sum_{j\in\mathbf{R}}(z_j-c_j)x_j$。

其中 $\mathbf{R}$ 是非基变量下标集，$z_j=\boldsymbol{c_B}\boldsymbol{B}^{-1}p_j$。

单纯形方法计算步骤如下：

首先给定一个初始基本可行解，设初始基为 $\boldsymbol{B}$，然后执行下列主要步骤：

（1）解 $\boldsymbol{B}\boldsymbol{x_B}=\boldsymbol{b}$，求得 $\boldsymbol{x_B}=\boldsymbol{B}^{-1}\boldsymbol{b}=\boldsymbol{b}$，令 $\boldsymbol{x_N}=0$，计算目标函数值 $f=\boldsymbol{c_B}\boldsymbol{x_B}$。

（2）求单纯形乘子 w，解 $w\boldsymbol{B}=\boldsymbol{c_B}$。对于所有非基变量，计算判别数 $z_j-c_j=w_jp_j-c_j$。令 $z_k-c_k=\max\limits_{j\in\mathbf{R}}\{z_j-c_j\}$，若 $z_k-c_k\leqslant 0$，则对于所有非基变量 $z_j-c_j\leqslant 0$，对应基变量的判别数总是为零，因此停止计算，现行基本可行解是最优解。否则，进行下一步。

（3）解 $\boldsymbol{B}y_k=p_k$，得到 $y_k=\boldsymbol{B}^{-1}p_k$，若 $y_k\leqslant 0$，即 y_k 的每个分量均非正数，则停止计算，

问题不存在有限最优解,否则进行步骤(4)。

(4) 确定下标 r,使

$$x_k = \frac{\overline{b_r}}{y_{rk}} = \min\left[\frac{\overline{b_i}}{y_{ik}}\right] y_{ik} > 0$$

其中,x_{B_r} 为离基变量,x_k 为进基变量。用 p_k 替格 p_{B_r},得到新的基矩阵 $\boldsymbol{B}$,返回步骤(1)。

【例 8-3】 用单纯形方法解下列问题:

$$\min f(x) = x_1 - 2x_2 + x_3$$

$$\text{s. t.} \begin{cases} x_1 + x_2 - 2x_3 + x_4 = 10 \\ 2x_1 - x_2 + 4x_3 \leqslant 8 \\ -x_1 + 2x_2 - x_3 \leqslant 4 \\ x_j \geqslant 0, j = 1,2,3,4 \end{cases}$$

解:根据单纯形方法计算步骤,编写算法代码如下:

```
function [x,f] = Dmin(c,A,b,AR,y0,d)
%x: 最优解
%f: 目标函数最优值
%c: 目标函数系数向量
%A: 系数矩阵
%b: m 维列向量
%AR: 松弛变量系数矩阵
%y0: 基矩阵初始向量
%d: 补充向量(非目标系数向量, 为一零向量)
N = 10000;
B = [A,AR,b];
[m,n] = size(B);
C = [c,d];
y = y0;
x = zeros(1,length(c));
for k = 1:N
    k;
    z = B(:,end);              %右端
    for j = 1:n-1
        t(j) = y * B(:,j) - C(j);   %检验数
    end
    t;
    f = y * z;

     %选取主列%
  [alpha,q] = max(t);
    q;
    W(k) = q;                  %x下标矩阵
     %选取主元
    for p = 1:m
        if B(p,q)<= 0
            r(p) = N;
        else r(p) = z(p)/B(p,q);
        end
```

```
    end


    [beta,p] = min(r);
    p;
    y(p) = C(q)

    B(p,:) = B(p,:)/B(p,q);
    for i = 1:m
        if i~ = p
            B(i,:) = B(i,:) - B(p,:) * B(i,q);
        end
    end
    if max(t)<= 0
        break;
    end
    B
end

Z = B(:,end)
if length(x(W))~ = length(Z)
    x = char('NONE');
f = char(' NONE');
disp('不存在有限最优解');
else x(W) = Z';
end
```

根据题意，编写主函数代码如下：

```
clear all
clc
c = [1 -2 1];
A = [1 1 -2 1;
    2 -1 4 0;
    -1 2 -4 0];
b = [10;8;4];
AR = [0 0;1 0;0 1];
y0 = [0 0 0];
d = [0 0 0];
[x,f] = Dmin(c,A,b,AR,y0,d)
```

运行后得到结果如下：

```
y =
     0     0    -2
B =
    1.5000          0          0    1.0000          0   -0.5000    8.0000
    1.5000          0     2.0000         0     1.0000    0.5000   10.0000
   -0.5000     1.0000    -2.0000         0          0    0.5000    2.0000
```

```
y =
     0     1    -2
B =
   1.5000         0         0    1.0000         0   -0.5000    8.0000
   0.7500         0    1.0000         0    0.5000    0.2500    5.0000
   1.0000    1.0000         0         0    1.0000    1.0000   12.0000
y =
     0     1    -2
Z =
     8
     5
    12
x =
     0    12     5
f =
   -19
```

在单纯形法求解过程中，每一个基本可行解 $\boldsymbol{x}$ 都以某个经过初等行变换的约束方程组中的单位矩阵为可行基。对于极大化的线性规划问题，要先标准化，即将极大化问题变换为极小化问题，然后再利用单纯形方法求解。

8.1.3 Powell 法

Powell 法是一种有效的直接搜索法，这种方法本质上是共轭方向法。Powell 法把整个计算过程分成若干个阶段，每一阶段（一轮迭代）由 $n+1$ 次一维搜索组成。在算法的每一阶段中，先依次沿着已知的 n 个方向搜索，得一个最好点，然后沿着本阶段的初点与该最好点连线方向进行搜索，求得这一阶段的最好点。再利用最后的搜索方向取代前 n 个方向之一，开始下一阶段的迭代。

【例 8-4】 编程求解函数 $f(x)=x_1^2+3x_2^2-4x_1-3x_1x_2$ 的极小点 x。初始点 $x_0=[1,1]^T$，迭代精度 $\varepsilon=0.001$。

解： 首先编写目标函数代码和一元函数最小值区间函数代码如下：

```
function m = y(x1,x2,d1,d2,alpha)          % 建立关于 α 的一元函数 y(α)
m = (x1 + alpha * d1)^2 + 2 * (x2 + alpha * d2)^2 - 4 * (x1 + alpha * d1) - 2 * (x1 + alpha * d1) *
(x2 + alpha * d2);

function [a,b] = section(x1,x2,d1,d2)      % 采用外推法求解一元函数的最小值区间
x11 = x1;x22 = x2;d11 = d1;d22 = d2;h0 = 1;h = h0;
alpha1 = 0;
y1 = y(x11,x22,d11,d22,alpha1);
alpha2 = h;
y2 = y (x11,x22,d11,d22,alpha2);
t = 0;
if y2 > y1
```

```
        h = -h;alpha3 = alpha1;y3 = y1;t = 1;
    end
    while(1)
        if t == 1
            alpha1 = alpha2;y1 = y2;
            alpha2 = alpha3;y2 = y3;
        else t = 1;
        end
        alpha3 = alpha2 + h;y3 = y(x11,x22,d11,d22,alpha3);
        if y3 < y2
            h = 2 * h;
        else
            break;
        end
    end
    if alpha1 > alpha3
        tem = alpha1;alpha1 = alpha3;alpha3 = tem;
        a = alpha1;b = alpha3;
    else a = alpha1;b = alpha3;
    end
```

在该优化设计过程中，黄金分割搜索法作为Powell算法主程序中的一部分。在Powell算法运行过程中会多次调用黄金分割搜索算法程序。这样可以缩短优化设计计算时间。

黄金分割搜索法函数MATLAB代码如下：

```
    function alpha = ALPHA(x1,x2,d1,d2,A,B) % 利用黄金分割法求解关于α的函数y(α)的极小点α*
    x11 = x1;x22 = x2;d11 = d1;d22 = d2;
    a = A;b = B;
    ep = 0.001;r = 0.618;
    alpha1 = b - r * (b - a);
    y1 = y(x11,x22,d11,d22,alpha1);
    alpha2 = a + r * (b - a);
    y2 = y(x11,x22,d11,d22,alpha2);
    while(1)
        if y1 >= y2
            a = alpha1;alpha1 = alpha2;
            y1 = y2;
            alpha2 = a + r * (b - a);
            y2 = y(x11,x22,d11,d22,alpha2);
        else
            b = alpha2;alpha2 = alpha1;
            y2 = y1;
            alpha1 = b - r * (b - a);
            y1 = y(x11,x22,d11,d22,alpha1);
        end
        if abs(b - a)< ep && abs(y2 - y1)< ep
            break;
        end
    end
    alpha = 0.5 * (a + b);
```

Powell 算法程序 MATLAB 代码如下：

```
function [z,fmin] = powell(f) %在二维空间中求解 f(x1,x2)的最小值点,求解结果返回变量
                                        坐标(x1,x2)和极小值 fmin
k = 0;n = 2;
x = [0;0;];
ff(1) = f(x(1),x(2));
ep = 0.001;
d = [1;0;0;1];
while(1)
    x00 = [x(1);x(2);];
    for i = 1:n
        [a(i),b(i)] = section(x(2 * i - 1),x(2 * i),d(2 * i - 1),d(2 * i));
        alpha(i) = ALPHA(x(2 * i - 1),x(2 * i),d(2 * i - 1),d(2 * i),a(i),b(i));
        x(2 * i + 1) = x(2 * i - 1) + alpha(i) * d(2 * i - 1);
        x(2 * i + 2) = x(2 * i) + alpha(i) * d(2 * i);
        ff(i + 1) = f(x(2 * i + 1),x(2 * i + 2));
    end
    for i = 1:n
        Delta(i) = ff(i) - ff(i + 1);
    end
        delta = max(Delta);
        for i = 1:n
            if delta == Delta(i)
                m = i;
                break;
            end
        end
        d(2 * n + 1) = x(2 * n + 1) - x(1);
        d(2 * n + 2) = x(2 * n + 2) - x(2);
        x(2 * n + 3) = 2 * x(2 * n + 1) - x(1);
        x(2 * n + 4) = 2 * x(2 * n + 2) - x(2);
        ff(n + 2) = f(x(2 * n + 3),x(2 * n + 4));
        f0 = ff(1);f2 = ff(n + 1);f3 = ff(n + 2);
        k = k + 1;
        R(k,:) = [k,x',d',ff];
        if f3 < f0 && (f0 - 2 * f2 + f3) * (f0 - f2 - delta)^2 < 0.5 * delta * (f0 - f3)^2
        [a(n + 1),b(n + 1)] = section(x(2 * n + 1),x(2 * n + 2),d(2 * n + 1),d(2 * n + 2));
        alpha(n + 1) = ALPHA(x(2 * n + 1),x(2 * n + 2),d(2 * n + 1),d(2 * n + 2),a(n + 1),b(n + 1));
        x(1) = x(2 * n + 1) + alpha(n + 1) * d(2 * n + 1);
        x(2) = x(2 * n + 2) + alpha(n + 1) * d(2 * n + 2);
        for i = m:n
            d(2 * i - 1) = d(2 * i + 1);
            d(2 * i) = d(2 * i + 2);
        end
        else
         if f2 < f3
             x(1) = x(2 * n + 1);
             x(2) = x(2 * n + 2);
```

```
            else
                x(1) = x(2 * n + 3);
                x(2) = x(2 * n + 4);
            end
          end
      RR(k,:) = alpha;
      ff(1) = f(x(1),x(2));
      if (((x(2 * n + 1) - x00(1))^2 + (x(2 * n + 2) - x00(2))^2)^(1/2))< ep
          break;
      end
  end
  z = [x(1);x(2)];
  fmin = f(x(1),x(2));
```

主函数代码如下：

```
clear all
clc
f = @(x1,x2) x1^2 + 3 * x2^2 - 4 * x1 - 3 * x1 * x2
[z,fmin] = powell(f)
```

运行后得到结果如下：

```
z =
    3.9998
    1.9998
fmin =
  - 11.9997
```

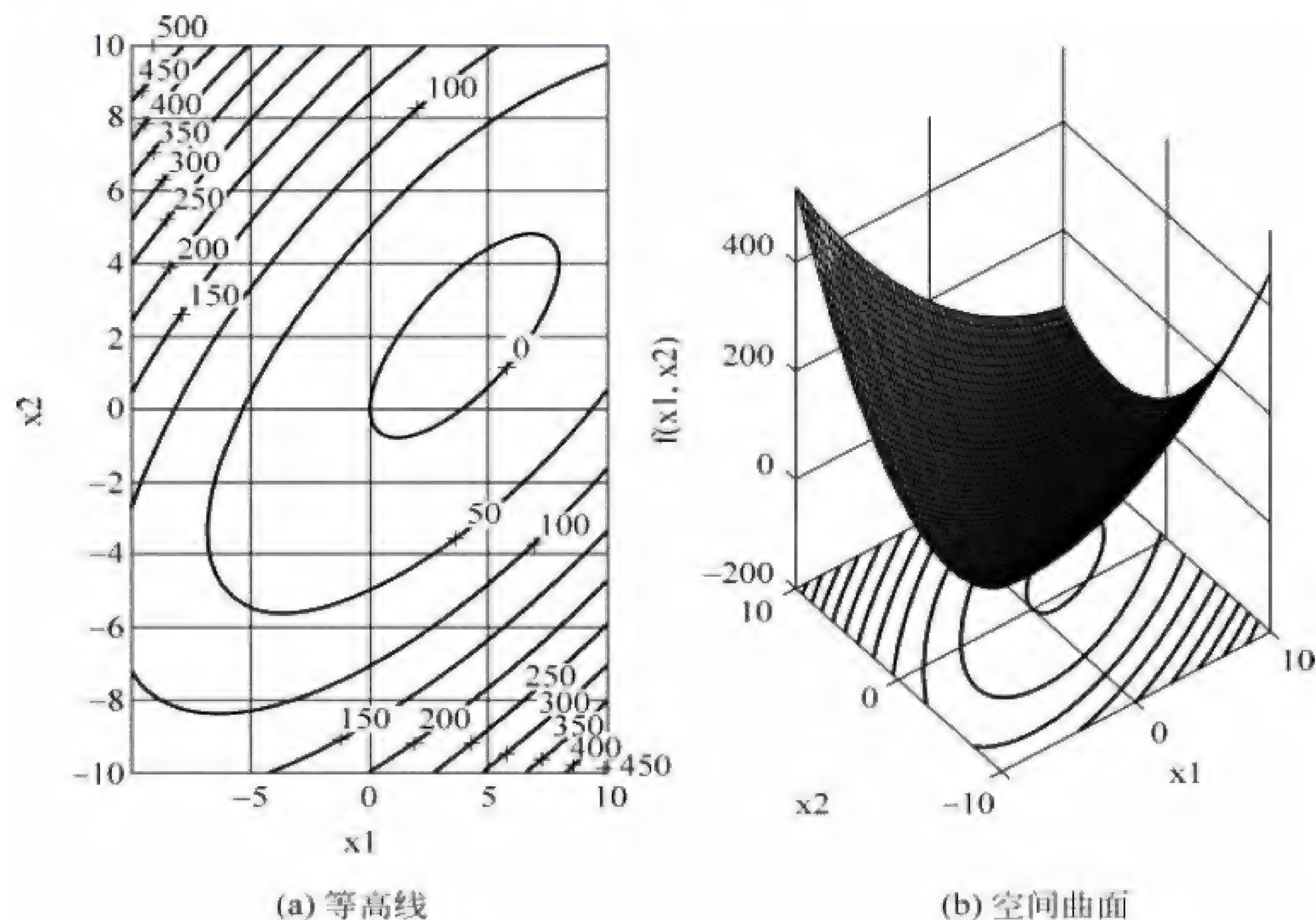
即极小值点为(3.9998,1.9998)，极小值为−11.9997。

同时，还可以通过 MATLAB 自带的 fminsearch 函数检验上述结果是否正确，其 MATLAB 代码如下：

```
x = linspace( - 10,10,50);
y = linspace( - 10,10,50);
[x,y] = meshgrid(x,y);
z = x.^2 + 2 * y.^2 - 4 * x - 2 * x. * y;
subplot(1,2,1);           % 在一幅框中显示两张图,显示第一张图
cs = contour(x,y,z);
clabel(cs);               % 绘制等高线图
xlabel('x1');
ylabel('x2');
title('(a)Contour plot');
grid;
subplot(1,2,2);           % 在一幅框中显示两张图,显示第二张图
cs = surfc(x,y,z);        % 绘制空间曲面图
```

```
zmin = floor(min(z));
zmax = ceil(max(z));
xlabel('x1');
ylabel('x2');
zlabel('f(x1,x2)');
title('(b) Mesh plot');
```

运行后得到等高线图和空间曲面图如图 8-2 所示：



(a) 等高线　　(b) 空间曲面

图 8-2　等高线图和空间曲面图

从图 8-2 中可以看出，函数的极小值点均在坐标(3.9998,1.9998)左右。

8.2　使用导数计算的间接法

8.2.1　最速下降法

最速下降法以负梯度方向作为最优化方法的下降方向，又称为梯度下降法，是最简单实用的方法。其基本思想为函数的负梯度方向是函数值在该点下降最快的方向。将 n 维问题转化为一系列沿负梯度方向用一维搜索方法寻优的问题，利用负梯度作为搜索方向，故称为最速下降法或梯度法。

由公式 $X^{(K+1)}=K^{(K)}+\alpha_K \vec{d}^{(K)}(k=0,1,2,\cdots)$可知，若某次迭代中已取得点 $X^{(k)}$，从该点出发，取负梯度方向$\vec{d}^{(k)}=-\dfrac{\nabla f(X^{(k)})}{\|\nabla f(X^{(k)})\|}$为搜索方向。则最速下降法的迭代公式为

$$X^{(K+1)}=X^{(K)}-\alpha_K\frac{\nabla f(X^{(k)})}{\|\nabla f(X^{(k)})\|}\quad(k=0,1,2,\cdots)$$

当第 k 次的迭代初始点 $X^{(k)}$ 和搜索方向 $\vec{d}^{(k)}$ 已经确定的情况下，原目标函数成为关于步长 α 的一维函数。即

$$\varphi(\alpha)=f(x^{(K)}+\alpha S^{(K)})$$

最优步长 α_K 可以利用一维搜索的方法求得，即

$$\min_{\alpha}\varphi(\alpha)=f(X^{(k+1)})=f(X^{(K)}+\alpha_k\vec{d}^{(k)})\ \min_{\alpha}f(X^{(K)}+\alpha\vec{d}^{(k)})$$

根据一元函数极值的必要条件和多元复合函数的求导公式，得

$$\varphi'(\alpha)=[\nabla f(X^{(K)}+\alpha\vec{d}^{(k)})]^{\mathrm{T}}\nabla f(X^{(K)})=0$$
$$[\nabla f(X^{(K+1)})]^{\mathrm{T}}\nabla f(X^{(K)})=0$$

或写成

$$[\vec{d}^{(K+1)}]^{\mathrm{T}}\vec{d}^{(k)}=0$$

由此可知，在最速下降法中相邻两个搜索方向互相正交。也就是说在用最速下降法迭代求优的过程中，走的是一条曲折的路线，该次搜索方向与前一次搜索方向垂直，形成“之”字形的锯齿现象。

最速下降法刚开始搜索步长比较大，越靠近极值点其步长越小，收敛速度越来越慢。特别是当二维二次目标函数的等值线是较扁的椭圆时，这种缺陷更加明显。因此所谓最速下降是指目标函数在迭代点附近出现的局部性质，从迭代过程的全局来看，负梯度方向并非是目标函数的最快搜索方向。

最速下降法理论明确、程序简单、对初始点要求不严格。对一般函数而言，最速下降法的收敛速度并不快，因为最速下降方向仅仅是指某点的一个局部性质。一般情况下，最速下降法与其他算法配合，在迭代开始时使用。

【例 8-5】 使用最速下降法，求解函数 $f(x)=x_1^2+2x_2^2$ 的极小值点。其中，误差为 0.0001，初始点为(1,1)。

解：编写最速下降法的 MATLAB 代码如下：

```
function x = fsxsteep(f,e,a,b)
%最速下降法
%x = fsxsteep(f,e,a,b)为输入函数 f 为允许误差 (a,b)为初始点;
x1 = a;x2 = b;
Q = fsxhesse(f,x1,x2);
x0 = [x1 x2]';
fx1 = diff(f,'x1');                                        %对 x1 求偏导数
fx2 = diff(f,'x2');                                        %对 x2 求偏导数
g = [fx1 fx2]';                                            %梯度
g1 = subs(g);                                              %把符号变量转为数值
d = -g1;
while (abs(norm(g1))>= e)
t = (-d)'*d/((-d)'*Q*d);t = (-d)'*d/((-d)'*Q*d);   %求搜索方向
x0 = x0 - t*g1;                                            %搜索到的点
v = x0;
a = [1 0]*x0;
b = [0 1]*x0;
x1 = a;
```

```
x2 = b;
g1 = subs(g);
d = - g1;
end;
x = v;

function x = fsxhesse(f,a,b)
% 求函数的 hesse 矩阵;
% x = fsxhesse(f)为输入函数 f 为二次函数 x1,x2 为自变量;
x1 = a;x2 = b;
fx = diff(f,'x1');            % 求 f 对 x1 偏导数
fy = diff(f,'x2');            % 求 f 对 x2 偏导数
fxx = diff(fx,'x1');          % 求二阶偏导数 对 x1 再对 x1
fxy = diff(fx,'x2');          % 求二阶偏导数 对 x1 再对 x2
fyx = diff(fy,'x1');          % 求二阶偏导数 对 x2 再对 x1
fyy = diff(fy,'x2');          % 求二阶偏导数 对 x2 再对 x2
fxx = subs(fxx);              % 将符号变量转化为数值
fxy = subs(fxy);
fyx = subs(fyx);
fyy = subs(fyy);
x = [fxx,fxy;fyx,fyy];        % 求 hesse 矩阵
```

主函数代码如下：

```
clear all
clc
syms x1 x2;
X = [x1,x2];
fx = X(1)^2 + 2 * X(2)^2;
x = fsxsteep(fx,0.001,1,1)
```

运行后得到结果如下：

```
x =
 81/614656
 81/614656
```

8.2.2 共轭梯度法

最速下降法具有计算简单、对初始点的选择要求低、最初几步迭代速度快的优点，但是越接近极值点时效果越差，在搜索过程中呈锯齿状的搜索路径。

设想在搜索过程中，如果能如图 8-3 所示，截弯取直在沿$\vec{d}^{(0)}$搜索到 $X^{(1)}$点后，不再沿 $X^{(1)}$的负梯度方向搜索，而改沿$\vec{d}^{(1)}$的方向进行搜索，由上述可知，任意形式的目标函数在极值点附近都近似于一个二次函数，希望对二次函数一次搜索就能够到达极小值点 X^*。即有 $X^*=X^{(1)}+\alpha_1\vec{d}^{(1)}$。

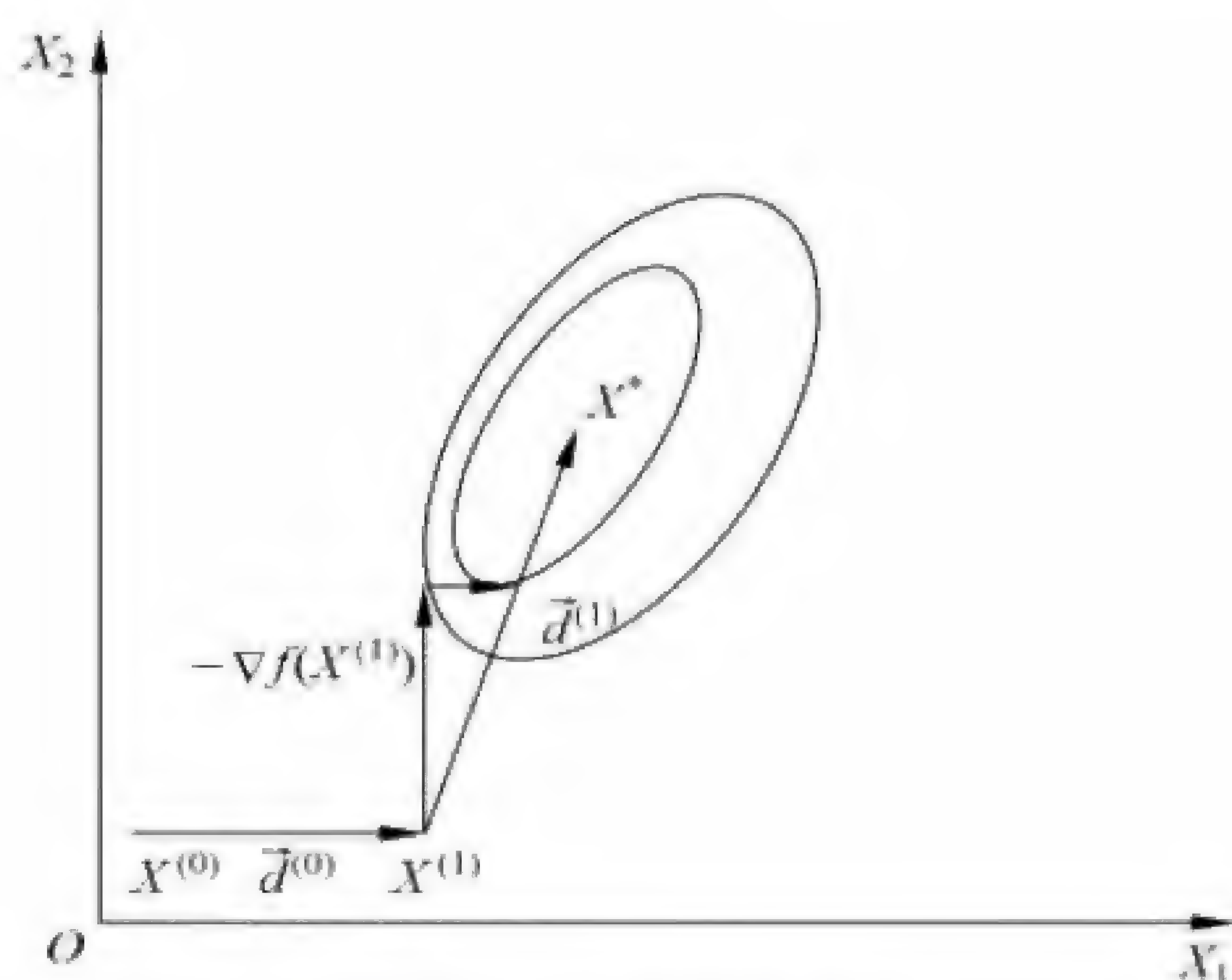


图 8-3 搜索示意图

其中，$\vec{d}^{(1)}$方向需要满足条件：

$$f(X) = \frac{1}{2}X^{\mathrm{T}}GX + B^{\mathrm{T}}X + C$$

在 $X^{(1)}$点的梯度为

$$\nabla f(X^{(1)}) = GX^{(1)} + B$$

因为 X^*为极小值点，应满足存在极值的必要条件，故有

$$\nabla f(X^*) = GX^* + B = 0$$

$$\nabla f(X^*) = G[X^{(1)} + \alpha_1 \vec{d}^{(1)}] + B$$

$$= \nabla f(X^{(1)}) + \alpha_1 G\vec{d}^{(1)} = 0$$

将等式两边同时左乘$[\vec{d}^{(0)}]^{\mathrm{T}}$。因为$\nabla f(X^{(1)})$与$\vec{d}^{(0)}$正交，$[\vec{d}^{(0)}]^{\mathrm{T}}\nabla f(X^{(1)})=0$。则有

$$[\vec{d}^{(0)}]^{\mathrm{T}}G\vec{d}^{(1)} = 0$$

这就是使$\vec{d}^{(1)}$直指极小值点 X^*所必须满足的条件。

两个向量$\vec{d}^{(0)}$和$\vec{d}^{(1)}$称为 $\boldsymbol{G}$ 的共轭向量，或称$\vec{d}^{(0)}$和$\vec{d}^{(1)}$对 $\boldsymbol{G}$ 是共轭方向。可见，沿$\vec{d}^{(0)}$方向搜索得到 X^*后，再沿$\vec{d}^{(0)}$的共轭方向$\vec{d}^{(1)}$搜索就可以到达极小值点 X^*。

共轭梯度法的迭代步骤如下：

(1) 取初始点 $X^{(0)}$，计算精度 ε；

(2) 令 $k=0$，$\vec{d}^{(k)}=\nabla f(X^{(k)})$；

(3) 从 $X^{(k)}$出发沿$\vec{d}^{(k)}$方向一维搜索到 $X^{(k+1)}$；

(4) 计算梯度$\nabla f(X^{(k+1)})$，检验是否满足 $\|\nabla f(X^{(k+1)})\| \leqslant \varepsilon$，若满足则迭代停止，$X^{(k+1)}$点为极小值点，否则进行步骤(5)；

(5) 判断 $k=n$ 是否成立，若 $k=n$，则令 $X^{(0)} \Leftarrow X^{(k+1)}$，返回步骤(2)，否则计算 $\beta_k = \frac{\|\nabla f(X^{(k+1)})\|^2}{\|\nabla f(X^{(k)})\|^2}$、$\vec{d}^{(k+1)} = -\nabla f(X^{(k+1)}) + \beta_k \vec{d}^{(k)}$，令 $k=k+1$，返回步骤(3)。

【例 8-6】 用共轭梯度法求二次函数 $f(x_1,x_2)=x_1^2+x_2^2-5x_1-3x_1x_2$ 的极小值点及极小值。其中，初始点为[1,0]，迭代精度 $\varepsilon=0.0001$。

解：编写共轭梯度法求解函数极值的 MATLAB 函数代码如下：

```
%% 共轭梯度函数 %%
function f = Conjugate(x0,t)
% 初始点坐标：x0
% 收敛精度：t
% 求得函数的极值：f
x = x0;
syms xi yi a;                              % 定义自变量，步长为符号变量
f = xi^2 + yi^2 - 5 * xi - 3 * xi * yi;    % 创建符号表达式 f
% 求表达式 f 对 xi、yi 的一阶求导
fx = diff(f,xi);
fy = diff(f,yi);
% 代入初始点坐标计算对 xi、yi 的一阶求导实值
fx = subs(fx,{xi,yi},x0);
fy = subs(fy,{xi,yi},x0);
fi = [fx,fy];                              % 初始点梯度向量
count = 0;                                 % 搜索次数初始为 0
while double(sqrt(fx^2 + fy^2))> t         % 搜索精度不满足已知条件
    s = - fi;                              % 第一次搜索的方向为负梯度方向
    if count <= 0
        s = - fi;
    else
        s = s1;
    end
    x = x + a * s;
    f = subs(f,{xi,yi},x);
    f1 = diff(f);
    f1 = solve(f1);
    if f1~ = 0
        ai = double(f1);                   % 强制转换数据类型为双精度数值
    else
        break                              % 若 a = 0，则直接跳出循环，此点即为极值点
    end
    x = subs(x,a,ai);                      % 得到一次搜索后的点坐标值
    f = xi^2 + yi^2 - 5 * xi - 3 * xi * yi;
    fxi = diff(f,xi);
    fyi = diff(f,yi);
    fxi = subs(fxi,{xi,yi},x);
    fyi = subs(fyi,{xi,yi},x);
    fii = [fxi,fyi];
    d = (fxi^2 + fyi^2)/(fx^2 + fy^2);
    s1 = - fii + d * s;
    count = count + 1;
    fx = fxi;
    fy = fyi;                              % 搜索后终点坐标变为下一次搜索的始点坐标
end
x                                          % 输出极值点
f = subs(f,{xi,yi},x)                      % 输出极小值
count                                      % 输出搜索次数
```

已知初始点为[1,0]，迭代精度 $\varepsilon=0.0001$，求解函数极值命令代码如下：

```
clear all
clc
f = Conjugate([1,0],0.0001)
```

运行后得到结果如下：

```
x =
[ -2, -3]
f =
5
count =
     1
f =
  5
```

即经过 1 次迭代，极小值点坐标为[－2，－3]，极小点值为 5。

8.3 拟牛顿法

牛顿法在实际应用中需要存储二阶导数信息和计算一个矩阵的逆矩阵，这对计算机的时间和空间要求都比较高，也容易遇到不正定的 Hessian 矩阵和病态的 Hessian 矩阵，导致求出来的逆很古怪，从而使算法沿着一个不理想的方向去迭代。

牛顿法成功的关键是利用了 Hessian 矩阵提供的曲率信息。但是计算 Hessian 矩阵工作量大，并且有些目标函数的 Hessian 矩阵很难计算，甚至不好求出，这就使得仅利用目标函数的一阶导数的方法更受欢迎。

拟牛顿法就是利用目标函数值 f 和一阶导数 g（梯度）的信息，构造出目标函数的曲率近似，而不需要明显形成 Hessian 矩阵，同时具有收敛速度快的特点。它是基于牛顿法，并对其进行了重大改进的一种方法。

拟牛顿法的计算步骤如下：

(1) 取初始点 $X^{(0)}$，计算精度 ε；

(2) 令 $k=0$，$\boldsymbol{A}_0=\boldsymbol{I}$（单位矩阵）；

(3) 计算 $X^{(k)}$ 点的梯度 $\nabla f(X^{(k)})$ 及其模 $\|\nabla f(X^{(k)})\|$；

(4) 判断是否满足精度指标 $\|\nabla f(X^{(k)})\|\leqslant\varepsilon$，若满足，$X^{(k)}$ 为最优点，迭代停止，输出最优解 $X^{*}=X^{(k)}$ 和 $f(X^{*})=f(X^{(k)})$，否则进行步骤(5)的计算；

(5) 构造搜索方向 $\vec{d}^{(k)}=-A_K\nabla f(X^{(k)})$ 进行一维搜索，求最优步长 α_K，并求出新点 $\min\limits_{\alpha} f(X^{(K)}+\alpha\vec{d}^{(k)})=f(X^{(K)}+\alpha_K\vec{d}^{(k)})$，$X^{(k+1)}=X^{(k)}+\alpha_K\vec{d}^{(K)}$，令 $k=k+1$，返回步骤(3)；

(6) 按尺度变换公式计算 $\Delta\boldsymbol{A}_k$ 及 $\boldsymbol{A}_k$；

(7) 令 $k=k+1$，返回步骤(3)。

【例 8-7】 用牛顿法求非线性方程组

$$\begin{cases} x_1 - 0.3\cos x_2 = 0 \\ x_2 - 0.5\sin x_1 = 0 \end{cases}$$

其中,起始点为(1,0),误差为0.01。

解:编写拟牛顿函数的 MATLAB 代码如下:

```
%拟牛顿法函数
function [Z,P,k,e] = newton(P,e0)
%用 P 输入初始猜想矩阵,不断迭代输出计算解
%Z 为迭代结束后的 F 矩阵
%k 为迭代次数,
%e 为每次迭代后的无穷范数,
%e0 为误差限
Z = Fun(P(1),P(2));
J = JFun(P(1),P(2));
Q = P - J\Z;
e = norm((Q - P),inf);
P = Q;
Z = Fun(P(1),P(2));
k = 1;
while e >= e0
    J = JFun(P(1),P(2));
    Q = P - J\Z;
    e = norm((Q - P),inf);
    P = Q;
    Z = Fun(P(1),P(2));
    k = k + 1;
end
end

%计算每一步的 F(x)
function [out] = Fun(x,y)
syms x1 x2;
f1 = x1 - 0.5 * cos(x2);
f2 = x2 - 0.5 * sin(x1);
Y = [f1;f2];
x1 = x;
x2 = y;
out = subs(Y);
end

%用来求每一步的 Jacobi 矩阵
function [y] = JFun(x,y)
syms x1 x2
f1 = x1 - 0.3 * cos(x2);
f2 = x2 - 0.5 * sin(x1);
df1x = diff(sym(f1),'x1');
df1y = diff(sym(f1),'x2');
```

```
df2x = diff(sym(f2),'x1');
df2y = diff(sym(f2),'x2');
j = [df1x,df1y;df2x,df2y];
%j中的元素为一阶偏导数
x1 = x;
x2 = y;
y = subs(j);
end
```

用牛顿法求非线性方程组的代码如下：

```
clear all
clc
P = [1 0]';
e0 = 0.01;
[Z P k e] = newton(P,e0)
```

运行后得到结果如下：

```
Z =
    6.667870800242115e-05
    1.609304681106372e-06
P =
    0.296863475587250
    0.146262770166111
k =
    3
e =
    0.004667380202889
```

上述结果中，P 为计算解，k 为迭代次数，Z 为第 k 次迭代后的 F 矩阵，e 为第 k 次迭代后的误差。

本章小结

本章介绍了多种无约束优化方法，这些方法又分为间接法和直接法两类。间接法需要求目标函数的导数，又称为导数类方法。仅用到目标函数一阶导数的方法称为一阶方法，如梯度法和共轭梯度法；需要用到目标函数的二阶导数的方法称为二阶方法，如牛顿法。直接法不需要求目标函数的导数，因此又称为零阶方法，如坐标轮换法、鲍威尔法和单形替换法。评价这些算法应从以下三个方面来考察：

1. 可靠性

方法的可靠性是指在一定的精度要求下，求解各种问题的成功率。显然能求解出的问题越多，算法的可靠性就越好，通用性越强。

2. 有效性

有效性是指各方法的解题效率。即对同一题目，在相同的精度要求和初始条件下所需要计算函数的次数以及花费的时间。

3. 简便性

方法的简便性是指用这一算法解题的难易程度。包括编制程序的复杂程度，计算中需要调整的参数的多少，以及实现这一算法对计算机的要求，如存储空间大小等。

就可靠性而言，牛顿法最差。就有效性而言，坐标轮换法、单形替换法和最速下降法的计算效率较低，特别是对高维的优化问题和精度要求较高时更为明显。就简便性而言，牛顿法的程序编制比较复杂，此时牛顿法还需要占用较多的存储单元。当目标函数的一阶偏导数容易求时，使用最速下降法可使程序编制更加简单。一般来说，直接法都具有编程简单和所需存储单元少的优点。

从综合的效果来看，共轭梯度法具有较好的性能，因此目前应用最为广泛。

第9章 约束优化方法

约束优化问题是在自变量满足约束条件的情况下求目标函数最小化的问题，其中约束条件既可以是等式约束也可以是不等式约束。本章重点介绍随机方向法、复合形法、可行方向法和惩罚函数法四种典型的约束优化方法，并分别举例说明 MATLAB 在约束优化方法中的应用。

学习目标：

（1）了解约束优化方法；

（2）掌握 MATLAB 在不同约束优化方法中的应用。

9.1 约束优化方法简介

约束优化方法是寻求具有约束条件的线性或非线性规划问题解的数值算法。所谓约束优化问题，是指在约束条件之下求一点，该点称为最优解。约束优化问题中的函数均为凸函数时称为凸规划。凸函数有很多已知特性，因此凸规划算法的研究进展较快。线性规划是凸规划最简单的情形，可以用单纯形方法等求解。

将约束优化问题作为一个研究方向主要起源于以下两方面：

（1）大多数实际问题是包含约束条件的，这使得约束优化问题与实际问题息息相关。

（2）很多难于处理的问题（NP 难，或者 NP 完全等）是包含约束条件的，这使得约束优化问题在理论上非常具有挑战性。

约束优化问题的具体形式如下：

```
min f(x)
```

满足约束条件：

```
g(x)<=0
h(x) = 0
```

其中 x 是解向量，g(x)是不等式约束，h(x)是等式约束。

如果定义 F 为可行域，U 为非可行域，S 为搜索空间，则存在关

系——F 属于 S。一般来说，S 搜索空间包含两个非连通子集，可行域 F 和非可行域 U。

如果不等式 $g(x)$ 满足条件 $g(x)=0$，则这个约束条件称为点 x 的积极约束。任意一个等式约束条件都是可行域内所有点的积极约束。

9.2 随机方向法

随机方向法是一种原理简单的直接解法。它的基本思路是在可行域内选择一个起始点，利用随机数的概率特性，产生若干个随机方向，并从中选择一个能够使目标函数值下降最快的随机方向作为可行搜索方向，记做 $\vec{d}$。

从初始点 x_0 出发，沿 $\vec{d}$ 方向以一定的步长进行搜索，得到新点 x，新点 x 应满足约束条件：

$$\begin{cases} g_j(x) \leqslant 0 (j=1,2,\cdots,m) \\ f(x) < f(x_0) \end{cases}$$

至此完成一次迭代。

然后，将起始点移至 x，即令 $x \rightarrow x_0$。重复以上过程，经过若干次迭代计算后，最终取得约束最优解。

随机方向法的优点是对目标函数的性态无特殊要求，可以编写程序计算，使用方便。由于可行搜索方法是从许多随机方向中选择的目标函数下降最快的方向，并且步长还可以灵活变动，所以此算法的收敛速度比较快，若能取得一个较好的初始点，迭代次数可以大大地减少。它是一种求解小型优化设计问题的十分有效的算法。

【例 9-1】 使用随机方向法，求解以下方程：

$$\min F(X) = (x_1-4)^2 + 5(x_2-7)^2$$

$$\text{s.t.} \begin{cases} g_1(x) = 48 - x_1^2 - x_2^2 \leqslant 0 \\ g_2(x) = x_2 - x_1 - 5 \leqslant 0 \\ g_3(x) = x_1 - 5 \leqslant 0 \end{cases}$$

解：根据题意编写 MATLAB 代码如下：

```
clear all
clc
x01 = 9;
x02 = 10;
%题中函数方程
f0 = ((x01 - 4).^2 + 5 * (x02 - 7).^2);
dir0 = rand(2,1);
dir0(1) = dir0(1) - 0.5;
dir0(2) = dir0(2) - 0.5;
recm = 100;
rm = 1;
a = 2;
prea = 0.001;
cis = 0.0001;
```

```
x11 = x01 + dir0(1) * a;
x12 = x02 + dir0(2) * a;
f1 = ((x11 - 5).^2 + 4 * (x12 - 6).^2);
test = abs(f1 - f0);
while a > prea
    a = a * 1/2;
    rm = 1;
while rm < recm
j = 0;
test = abs(f1 - f0);
%约束条件
yue1 = (48 - x11.^2 - x12.^2);
yue2 = x12 - x11 - 5;
yue3 = x11 - 5;
if f1 <= f0 && yue1 <= 0 && yue2 <= 0 && yue3 <= 0
    x01 = x11;
    x02 = x12;
    f0 = f1;
    j = 1;
    x11 = x01 + dir0(1) * a;
    x12 = x02 + dir0(2) * a;
    f1 = ((x11 - 5).^2 + 4 * (x12 - 6).^2);
end
if j == 0
    rm = rm + 1;
end
end
if test < cis && yue1 <= 0 && yue2 <= 0 && yue3 <= 0
    break;
    disp(test);
end
end
disp(x11);
disp(x12);
disp(f1);
```

运行程序得到结果如下：

```
8.6419
10.0633
79.3049
```

即当 $x_1=8.6419$，$x_2=10.0633$ 时，函数有最小值为 79.3049。

9.3 复合形法

复合形法的基本思路是在 n 维空间的可行域中选取 K 个设计点作为初始复合形（多面体）的顶点。然后比较复合形中各顶点目标函数的大小，其中目标函数值最大的点作

为坏点，以坏点之外其余各点的中心为映射中心，寻找坏点的映射点。一般说来此映射点的目标函数值总是小于坏点的，也就是说映射点优于坏点。以映射点替换坏点与原复合形除坏点之外其余各点构成 K 个顶点的新复合形。

如此反复迭代计算，在可行域中不断以目标函数值低的新点代替目标函数值最大的坏点从而构成新复合形，使复合形不断向最优点移动和收缩，直至收缩到复合形的各顶点与其形心非常接近，满足迭代精度要求时为止。

最后输出复合形各顶点中的目标函数值最小的顶点作为近似最优点。

【例 9-2】 使用复合形法求解以下方程：

$$\min F(X) = (x_1 - 5)^2 + 5(x_2 - 5)^2$$

$$\text{s.t.}\begin{cases} g_1(x) = 60 - x_1^2 - x_2^2 \leqslant 0 \\ g_2(x) = x_2 - x_1 - 5 \leqslant 0 \\ g_3(x) = x_1 - 5 \leqslant 0 \end{cases}$$

解：根据题意编写 MATLAB 代码如下：

```
clear all
clc
%四个初始点
x01(1) = 9;
x01(2) = 8;
x01(3) = 10;
x01(4) = 9.3;
x02(1) = 10;
x02(2) = 6;
x02(3) = 6;
x02(4) = 9.5;
%比较三个初始点函数值大小
f(1) = (x01(1) - 5).^2 + 5 * (x02(1) - 5).^2;
f(2) = (x01(2) - 5).^2 + 5 * (x02(2) - 5).^2;
f(3) = (x01(3) - 5).^2 + 5 * (x02(3) - 5).^2;
f(4) = (x01(4) - 5).^2 + 5 * (x02(4) - 5).^2;
%将函数最小的点放在数组第一个位置
%最大的点放在数组最后一个位置
for i = 1:4
    for j = 2:4
    if f(j - 1)> f(j)
        Q1 = x01(j - 1);
        Q2 = x02(j - 1);
        Qv = f(j - 1);
        x01(j - 1) = x01(j);
        x02(j - 1) = x02(j);
        f(j - 1) = f(j);
        x01(j) = Q1;
        x02(j) = Q2;
        f(j) = Qv;
    end
    end
```

```
end
cha = f(4) - f(1);
while cha > 0.05
%求形心点坐标,即去除最坏点
%最后一个点的坐标后,剩余点坐标的均值
xinx01 = (x01(1) + x01(2) + x01(3))/3;
xinx02 = (x02(1) + x02(2) + x02(3))/3;
%求反射心点坐标
ra = 1.3;
pra01 = xinx01 + ra * (xinx01 - x01(4));
pra02 = xinx02 + ra * (xinx02 - x02(4));
fpra = ((pra01 - 5).^2 + 4 * (pra02 - 6).^2);
yue1 = (60 - pra01.^2 - pra02.^2);
yue2 = pra02 - pra01 - 5;
yue3 = pra01 - 5;
if fpra <= f(4) && yue1 <= 0 && yue2 <= 0 && yue3 <= 0
    f(4) = fpra;
    x01(4) = pra01;
    x02(4) = pra02;
else
    ra = 0.1;
    pra01 = xinx01 + ra * (xinx01 - x01(4));
    pra02 = xinx02 + ra * (xinx02 - x02(4));
    fpra = ((pra01 - 5).^2 + 4 * (pra02 - 6).^2);
    f(4) = fpra;
    x01(4) = pra01;
    x02(4) = pra02;
end
%由小到大冒泡排序
for i = 1:4
    for j = 2:4
    if f(j - 1)> f(j)
        Q1 = x01(j - 1);
        Q2 = x02(j - 1);
        Qv = f(j - 1);
        x01(j - 1) = x01(j);
        x02(j - 1) = x02(j);
        f(j - 1) = f(j);
        x01(j) = Q1;
        x02(j) = Q2;
        f(j) = Qv;
    end
    end
end
%计算最大点与最小点的函数差
cha = f(4) - f(1)
end
x01
x02
disp(f)
```

运行程序结果如下：

```
cha =
  105.7400
cha =
    16
cha =
    6.8198
cha =
    4.0351
cha =
    3.1571
cha =
    2.3505
cha =
    1.4381
cha =
    1.2075
cha =
    0.7306
cha =
    0.5567
cha =
    0.3498
cha =
    0.2558
cha =
    0.1662
cha =
    0.1191
cha =
    0.0789
cha =
    0.0558
cha =
    0.0375
x01 =
    8.2835   8.2864   8.2879   8.2897
x02 =
    6.1242   6.1226   6.1216   6.1209
    10.8429   10.8607   10.8692   10.8804
```

即当 x_1 为 8.2835，x_2 为 6.1242，函数有最小值为 10.8429。

9.4 可行方向法

可行方向法，顾名思义，是一种始终在可行域内寻找下降方向的搜索法，以其收敛速度快、效果好的优点成为求解约束非线性问题的有代表性的直接解法，同时也是求解大

型约束优化问题的主要方法之一。

可行方向法是一类算法，可看作无约束下降算法的自然推广。典型策略是从可行点出发，沿着下降可行方向进行搜索，求出使目标函数值下降的新的可行点。

考虑只含线性约束的非线性规划问题

$$\min f(x)$$
$$\text{s. t.} \begin{cases} \boldsymbol{A}\boldsymbol{x} \geqslant \boldsymbol{b} \\ \boldsymbol{E}\boldsymbol{x} = \boldsymbol{e} \end{cases}$$

其中，$f(x)$为非线性函数，$\boldsymbol{A}\in \boldsymbol{R}^{m\times n}$，$\boldsymbol{E}\in \mathbf{R}^{l\times n}$，$\boldsymbol{b}\in \mathbf{R}^{m}$，$\boldsymbol{e}\in \mathbf{R}^{l}$。

线性约束规格保证了优化问题的可行方向集、线性化可行方向集以及序列化可行方向集是等同的。

当某个可行方向同时也是目标函数的下降方向时，沿此方向移动一定会在满足可行性的情况下改进迭代点的目标函数值。

目前已经提出许多可行方向法，用来处理具有线性约束的非线性规划问题。

可行性算法的流程图如图 9-1 所示。

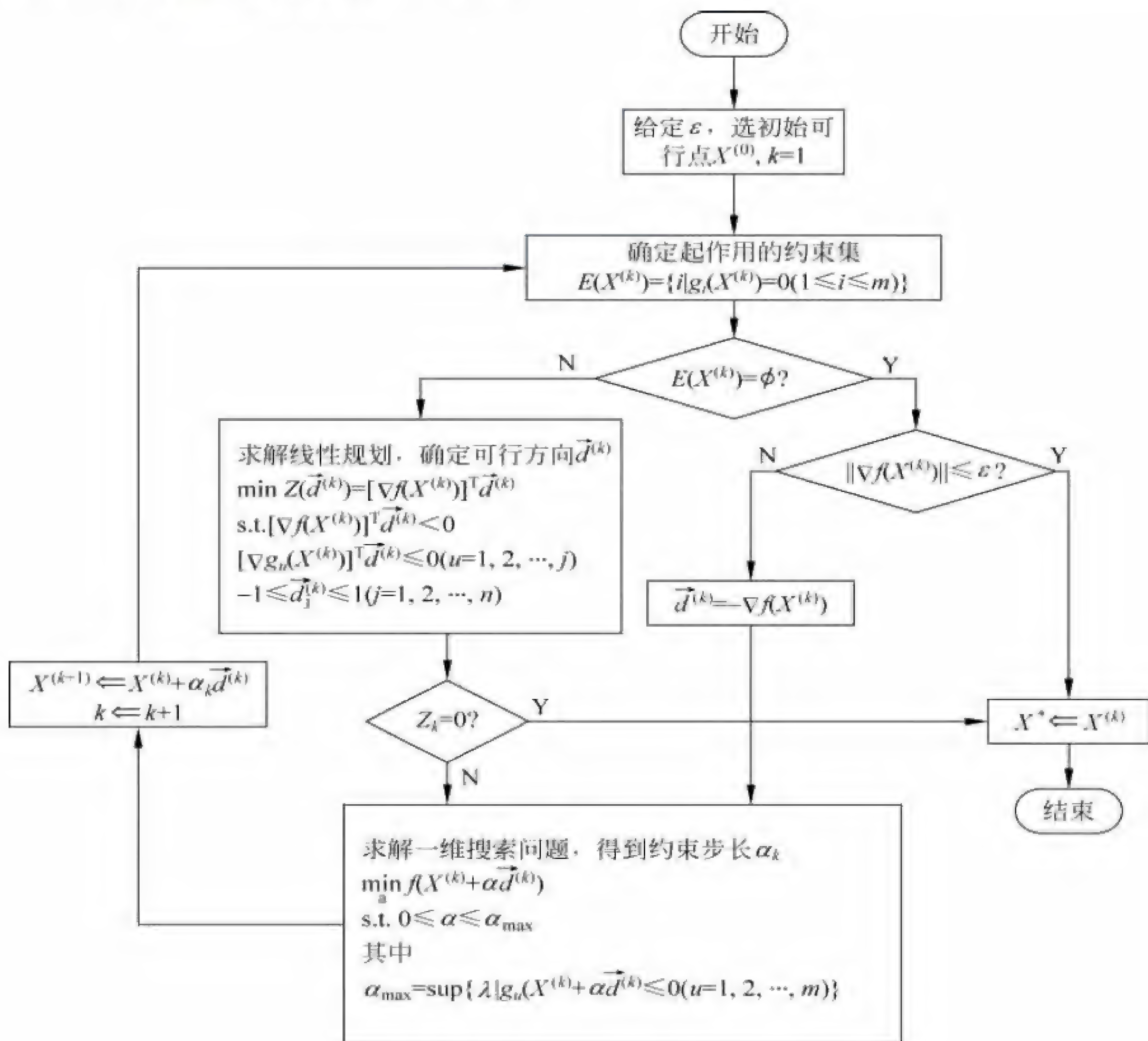


图 9-1　可行方向法流程图

【例 9-3】 使用复合形法求解以下方程：

$$\min F(X) = (x_1 - 1)^2 + (x_2 - 2)^2 + 1$$

$$\text{s.t.} \begin{cases} g_1(x) = 2x_1^2 - x_2^2 \leqslant 1 \\ g_2(x) = x_2 + x_1 \leqslant 2 \\ g_3(x) = -x_1 \leqslant 0 \end{cases}$$

解：根据题意编写 MATLAB 代码如下：

```
function kexingfangxiangfun
clc;
x0 = [0;0];
A = [2  -1;1 1;-1 0;0  -1];
b = [1;2;0;0];
c = 0;
kk = 0;
while c < 5
    c = c + 1;
    k = 0;j = 0;
    kk = kk + 1;
    fprintf('----kk = %d------',kk);
    A1 = [];b1 = [];
    A2 = [];b2 = [];
    for i = 1:4
        C = A(i,:) * x0;
        if C >= b(i) - 1e - 3
%不起作用约束
            k = k + 1;
            A1(k,:) = A(i,:);
            b1(k,1) = b(i);
        end
        if C < b(i) - 1e - 3
%起作用约束
            j = j + 1;
            A2(j,:) = A(i,:);
            b2(j,1) = b(i);
        end
    end
    A1,b1
    A2,b2
    if isempty(A1(:,:))
%A1 为 0 向量矩阵
        break
    end
    % pause
    f = dfxfun(x0);
    lb = [-1  -1];
    ub = [1 1];
    b0 = zeros(size(b1));
    d = linprog(f,A1,b0,[],[],lb,ub);
```

```
%求解最小化问题
    if (abs(d)<=1e-5)
        break
    end
    dd=A2*d;
    bb=b2-A2*x0;
    lamdmax=max(bb./dd);
    options=optimset('Display','off');
    lamda=fminbnd(@(lamda)fun(lamda,d,x0),0,lamdmax,options);
%求函数的局部极小值
    if (isempty(lamda(:)))
        break
    end
    x0=x0+lamda*d;
end
x0
fvall=fun1(x0);
x0=[0,1]';
[x,fval]=fmincon(@fun1,x0,A,b,[],[],[],[],[],options);
x
fval
end

%设置目标函数
function f= fun1(x)
f=x(1)^2+x(2)^2-2*x(1)-4*x(2)+6;
end

%目标函数
function f=fun(lamda,d,x)
xx=x+lamda*d;
f=xx(1)^2+xx(2)^2-2*xx(1)-4*xx(2)+6;
end

%目标函数求导后的函数
function dfx=dfxfun(x)
dfx=[2*x(1)-2,2*x(2)-4];
end
```

运行程序得到结果如下：

```
Optimization terminated.
x0 =
   0.499933898965784
   1.499933898965779
x =
   0.499999997884164
   1.499999982107960
fval =
   1.500000020007877
```

即当 x_1、x_2 分别为 0.499999997884164 和 1.499999982107960 时，函数有最小值为 1.500000020007877。

9.5 惩罚函数法

惩罚函数法的基本思想：利用原问题的目标函数和约束条件构造新的目标函数——罚函数，把约束最优化问题转化为相应罚函数的无约束最优化问题来求解。

考虑约束问题

$$\min f(x)$$
$$\text{s. t.} \begin{cases} g_i(x) \geqslant 0, & i = 1, \cdots, m \\ h_j(x) = 0, & j = 1, \cdots, l \end{cases}$$

求解的一种途径是由目标函数和约束函数组成辅助函数，把原来的约束问题转化为极小化辅助函数的无约束问题。

【例 9-4】 用惩罚函数外点法求解以下约束最优化问题：

$$\min f(X) = x_1 + x_2$$
$$\text{s. t.} \begin{cases} g_1(X) = x_1^2 - x_2 \leqslant 0 \\ g_2(X) = -x_1 \leqslant 0 \end{cases}$$

当惩罚因子分别为 5、10、50、100 的计算结果。

解：首先编写惩罚函数的 M 文件。

```
function f = CFfun(r0)
syms x1 x2
f = x1 + x2;
g1 = x1 ^2 - x2;
g2 = - x1;
r0 = 5;
c = 0.5;
km = 7;
k = 1:km;
r = r0 * c.^(k-1);
x1 = -1./(2 + 2. * r);
x2 = 1./(4. * (1 + r).^2) - 1./2. * r;
g1 = x1.^2 - x2;
g2 = - x1;
f = x1 + x2;
p = x1 + x2 + r. * g1.^2 + r. * g2.^2;
x1
x2
p % 惩罚函数
end
```

根据惩罚函数程序，运行以下代码：

```
clear all
clc
% 当惩罚因子为 5 时函数极小值
f5 = CFfun(5)
% 当惩罚因子为 10 时函数极小值
f10 = CFfun(10)
% 当惩罚因子为 50 时函数极小值
f50 = CFfun(50)
% 当惩罚因子为 100 时函数极小值
f100 = CFfun(100)
```

得到不同惩罚因子时的函数极小值：

```
x1 =
   -0.083333333333333   -0.142857142857143   -0.222222222222222   -0.307692307692308
   -0.380952380952381   -0.432432432432432   -0.463768115942029

x2 =
   -2.493055555555555   -1.229591836734694   -0.575617283950617   -0.217825443786982
   -0.011125283446712   0.108872808619430   0.176018365364419

p =
  28.708333333333332   2.584821428571429   -0.247829861111111   -0.405310997596154
 -0.339096795944940   -0.293387541899810   -0.270827348681464

f5 =
   -2.576388888888889   -1.372448979591837   -0.797839506172840   -0.525517751479290
   -0.392077664399093   -0.323559623813002   -0.287749750577610

x1 =
  1 至 6 列
   -0.083333333333333   -0.142857142857143   -0.222222222222222   -0.307692307692308
   -0.380952380952381   -0.432432432432432
  7 列
   -0.463768115942029

x2 =
  1 至 6 列
   -2.493055555555555   -1.229591836734694   -0.575617283950617   -0.217825443786982
   -0.011125283446712   0.108872808619430
  7 列
    0.176018365364419

p =
  1 至 6 列
  28.708333333333332   2.584821428571429   -0.247829861111111   -0.405310997596154
 -0.339096795944940   -0.293387541899810
  7 列
   -0.270827348681464
```

```
f10 =
  1 至 6 列
  -2.576388888888889   -1.372448979591837   -0.797839506172840   -0.525517751479290
  -0.392077664399093   -0.323559623813002
  7 列
  -0.287749750577610

x1 =
  1 至 6 列
  -0.083333333333333   -0.142857142857143   -0.222222222222222   -0.307692307692308
  -0.380952380952381   -0.432432432432432
  7 列
  -0.463768115942029
```

本章小结

最优化方法是应用数学的重要研究领域。它是研究在给定约束条件之下如何寻求某些因素(的量),以使某一(或某些)指标达到最优的一些学科的总称。简单来说,即以最优化数学模型来解决实际运用中的各种最优化问题。本章重点介绍了随机方向法、复合形法、可行方向法和惩罚函数法四种典型的约束优化方法。

第10章 二次规划

二次规划是非线性规划中的一类特殊数学规划问题，在很多方面都有应用，如投资组合、约束最小二乘问题的求解，序列二次规划在非线性优化问题中应用等。

在过去的几十年里，二次规划已经成为运筹学、经济数学、管理科学、系统分析和组合优化科学的基本方法。本章重点介绍二次规划的基本概念及二次规划包括的拉格朗日法和起作用集算法。

学习目标：

(1) 了解二次规划基本概念；

(2) 掌握使用 MATLAB 解决二次规划问题。

10.1 基本概念

二次规划问题是最简单的一类非线性约束优化问题，并且某些非线性规划问题可以转化为求解一系列二次规划问题，因此二次规划的求解方法也是求解非线性规划的基础之一。

如果某非线性规划的目标函数为自变量的二次函数，约束条件全是线性函数，就称这种规划为二次规划。其标准数学模型为

$$\min_{x} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}$$

$$\text{s. t.} \begin{cases} \boldsymbol{A} \cdot \boldsymbol{x} \leqslant \boldsymbol{b} \\ \boldsymbol{Aeq} \cdot \boldsymbol{x} = \boldsymbol{beq} \\ \boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub} \end{cases}$$

式中，$\boldsymbol{H}$，$\boldsymbol{A}$ 和 $\boldsymbol{Aeq}$ 为矩阵，$\boldsymbol{f}$，$\boldsymbol{b}$，$\boldsymbol{beq}$，$\boldsymbol{lb}$，$\boldsymbol{ub}$ 和 $\boldsymbol{x}$ 为列矢量。

其他形式的二次规划问题都可转化为标准形式。

在 MATLAB 中可以利用 quadprog 函数求解二次规划问题，其调用格式如下：

x=quadprog(H,f,A,b)：返回矢量 x，使函数 $\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}$ 最小化，其约束条件为 $\boldsymbol{A} \cdot \boldsymbol{x} \leqslant \boldsymbol{b}$；

x=quadprog(H,f,A,b,Aeq,beq)：仍然求解上面的问题，但添加了等式约束条件 $\boldsymbol{Aeq} \cdot \boldsymbol{x} \leqslant \boldsymbol{beq}$；

x=quadprog(H,f,A,b,lb,ub)：定义设计变量的下界 lb 和上界 ub，使得 $\boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub}$。

x=quadprog(H,f,A,b,lb,ub,x0)：同上，并设置初值 x0。

x=quadprog(H,f,A,b,lb,ub,x0,options)：根据 options 参数指定的优化参数进行最小化。

[x,fval]=quadprog(…)：返回解 x 和 x 处的目标函数值 fval。

[x,fval,exitflag]=quadprog(…)：返回 exitflag 参数，描述计算的退出条件。

[x,fval,exitflag,output]=quadprog(…)：返回包含优化信息的结构输出 output。

[x,fval,exitflag,output,lambda]=quadprog(…)：返回解 x 处包含拉格朗日乘子的 lambda 结构参数。

【例 10-1】 求解下面的最优化问题：

目标函数为

$$f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 3x_1 - 5x_2$$

约束条件为

$$\begin{cases} x_1 + x_2 \leqslant 2 \\ -x_1 + 2x_2 \leqslant 2 \\ 2x_1 + x_2 \leqslant 3 \\ x_1 \geqslant 0, x_2 \geqslant 0 \end{cases}$$

解：目标函数可以修改为

$$\begin{aligned} f(x) &= \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 3x_1 - 5x_2 \\ &= \frac{1}{2}(x_1^2 - 2x_1x_2 + 2x_2^2) - 3x_1 - 5x_2 \end{aligned}$$

记 $\boldsymbol{H} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}, \boldsymbol{f} = \begin{pmatrix} -3 \\ -5 \end{pmatrix}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ -1 & 2 \\ 2 & 1 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$

则上面的优化问题可写为

$$\min_x \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}$$

$$\text{s. t.} \begin{cases} \boldsymbol{A} \cdot \boldsymbol{x} \leqslant \boldsymbol{b} \\ (0\ 0)^{\mathrm{T}} \leqslant \boldsymbol{x} \end{cases}$$

编写 MATLAB 程序代码如下：

```
clear all
clc
H = [1 -1; -1 2];
f = [ -3; -5];
A = [1 1; -1 2;2 1];b = [2;2;3];
lb = zeros(2,1);
[x,fval,exitflag] = quadprog(H,f,A,b,[],[],lb)
```

运行结果如下：

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non - decreasing in
feasible directions, to within the default value of the optimality tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.

<stopping criteria details>

x =
    0.800000000000000
    1.200000000000000
fval =
   - 7.600000000000000
exitflag =
      1
```

10.2 拉格朗日法

拉格朗日法又称随体法：跟随流体质点的运动，记录该质点在运动过程中物理量随时间变化规律。

拉格朗日法是以研究单个流体质点运动过程作为基础，综合所有质点的运动，构成整个流体的运动。

以某一起始时刻每个质点的坐标位置(a,b,c)，作为该质点的标志。

任何时刻任意质点在空间的位置(x,y,z)都可以看成是(a,b,c)和t的函数。

拉格朗日法基本特点：追踪流体质点的运动。

优点：可直接运用固体力学中质点动力学进行分析。

基本的拉格朗日乘子法就是求函数$f(x_1,x_2,\cdots)$在$g(x_1,x_2,\cdots)=0$的约束条件下的极值的方法。其主要思想是将约束条件函数与原函数联系到一起，使其能配成与变量数量相等的等式方程，从而求出原函数极值的各个变量的解。

拉格朗日乘子法主要包含 PH 算法和 PHR 算法。

1. PH 算法(约数为等式的情况引入)

效用函数为

$$\min M(x,\boldsymbol{u}^{(k)},\sigma_k) = f(x) + \boldsymbol{u}^{(k)\mathrm{T}}h(x) + \sigma_k h(x)^{\mathrm{T}}h(x)$$

判断函数为

$$\phi_k = \| h(x^{(k)}) \|$$

当$\phi_k=\phi(x^{(k)})<\varepsilon$时迭代停止。

步骤(1)：选定初始点$x^{(0)}$，初始拉格朗日乘子向量$u^{(1)}$，初始罚因子σ_1及其放大系数$c>1$，控制误差$\varepsilon>0$与常数$\theta\in(0,1)$，令$k=1$。

步骤(2)：以$x^{(k+1)}$为初始点，求解无约束问题：

$$\min M(x,\boldsymbol{u}^{(k)},\sigma_k) = f(x) + \boldsymbol{u}^{(k)\mathrm{T}}h(x) + \sigma_k h(x)^{\mathrm{T}}h(x)$$

得到无约束问题最优解 $x^{(k)}$。

步骤(3)：当 $\| h(x^{(k)}) \| < \varepsilon$ 时，$x^{(k)}$ 为所求的最优解，停止迭代；否则转步骤(4)。

步骤(4)：当 $h(x^{(k)}) / \| h(x^{(k)}) \| < \theta$ 时，转步骤(5)；否则令 $\sigma_{k+1} = c\sigma_k$，转步骤(5)。

步骤(5)：令 $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \sigma_k h(x^{(k)})$，$k = k+1$，转步骤(1)。

2. PHR 算法(一般约束形式的松弛变量法和指数形式法)

松弛变量法：

$$M(u,v,\rho) = f(x) + \frac{1}{2\rho}\sum_{2\rho}^{1}\{[\max(0,u_i + \rho g_i(x))]^2 - u_i^2\} + \sum_{j=1}^{l} v_j h_j(x) + \frac{\rho}{2}\sum_{j=1}^{l} h_j^2(x)$$

乘子的修正公式为

$$v_j^{(k+1)} = v_j^{(k)} + \rho h_j(x^{(k)}), \quad j = 1,\cdots,l$$

$$u_i^{(k+1)} = \max[0, u_i^{(k)} + \rho g_i(x^{(k)})], \quad i = 1,\cdots,m$$

判断函数为

$$\phi_k = \left\{ \sum_{j=1}^{l} h_j^2(x^{(k)}) + \sum_{i=1}^{m} \max\left(-g_i(x^{(k)}), \frac{u_i^{(k)}}{\rho}\right)^2 \right\}^{1/2}$$

当 $\phi_k = \phi(x^{(k)}) < \varepsilon$ 时迭代停止。

【例 10-2】 采用拉格朗日乘子法求最优化问题如下：

$$\min l(x,\lambda) = x_1 - x_2 + \lambda(x_1^2 + x_1^2)$$

解：在 MATLAB 中编写代码如下：

```
clear all
clc
x = zeros(1,2)
% 用 syms 表示出转化后的无约束函数
syms x y lama
f = x - y + lama * (x^2 + y^2 - 2);
% 分别求函数关于 x、y、lama 的偏导
dx = diff(f,x);
dy = diff(f,y);
dlama = diff(f,lama);
% 令偏导为零,求解 x、y
xx = solve(dx,x);
% 将 x 表示为 lama 函数
yy = solve(dy,y);
% 将 y 表示为 lama 函数
ff = subs(dlama,{x,y},{xx,yy});
% 代入 dlama 得关于 lama 的一元函数
lamao = solve(ff);
% 求解得 lama0
xo = subs(xx,lama,lamao)
% 求得取极值处的 x0
yo = subs(yy,lama,lamao)
```

```
%取极值处的 y0
fo = subs(f,{x,y,lama},{xo,yo,lamao})
%取极值处的函数值
```

运行后得到结果如下：

```
x =
     0   0
xo =
     1
   - 1
yo =
   - 1
     1
fo =
     2
   - 2
```

10.3 起作用集算法

运用起作用集法，在每次迭代中，以已知的可行点为起点，把在该点起作用的约束作为等式约束，在此约束下极小化目标函数，其余的约束暂时不予考虑，求得新的比较好的可行点之后，再重复上述过程。这样，就将一般约束的二次规划问题转化为有限个仅带有等式约束的二次规划问题。考虑具体不等式约束的二次规划问题如下：

$$\min q(x) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{G}\boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{c}$$

$$\text{s.t.}\ \boldsymbol{A}\boldsymbol{x} \geqslant \boldsymbol{b}$$

其中，$\boldsymbol{G}$ 是 n 阶对称正定矩阵，$\boldsymbol{c}$ 是 n 维列向量，$\boldsymbol{A}$ 是 $m \times n$ 阶矩阵，$\boldsymbol{b}$ 是 m 维列向量，$\boldsymbol{x} \in \mathbf{R}^n$。

设在第 k 次迭代中，已知可行点 $\boldsymbol{x}_k$，在该点起作用约束指标集用 w_k 表示，将不等式约束问题转化为如下等式约束问题：

$$\min q(x) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{G}\boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}} - \boldsymbol{c}$$

$$\text{s.t.}\ a_i^{\mathrm{T}}\boldsymbol{x} = b_i, \quad i \in w_k$$

其中 a_i 是矩阵 $\boldsymbol{A}$ 的第 i 行，也是 $\boldsymbol{x}_k$ 处的起作用约束函数梯度。

现将坐标原点移到 $\boldsymbol{x}_k$ 处，令 $\boldsymbol{p} = \boldsymbol{x} - \boldsymbol{x}_k$，$\boldsymbol{g}_k = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$

将 $\boldsymbol{x}$ 代入目标函数得：$q(x) = q(\boldsymbol{x}_k + p) = \frac{1}{2}\boldsymbol{p}^{\mathrm{T}}\boldsymbol{G}\boldsymbol{p} + \boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{p} + \boldsymbol{p}_k$

其中，$\boldsymbol{p}_k = \frac{1}{2}\boldsymbol{x}_k^{\mathrm{T}}\boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_k$ 为常数项，不影响最优解，从而可以去掉。于是等式约束问题可以转化为求校正量 $\boldsymbol{p}_k$ 的问题：

$$\min\left\{\frac{1}{2}\boldsymbol{p}^{\mathrm{T}}\boldsymbol{G}\boldsymbol{p}^{\mathrm{T}} + \boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{p}\right\}$$

$$\text{s.t.}\ a_i^{\mathrm{T}}\boldsymbol{p}=0,\quad i\in w_k$$

解二次规划，求出最优解 $\boldsymbol{p}_k$。

【例 10-3】 用起作用集方法求解二次规划问题如下：

$$\min f(x)=x_1^2-x_1x_2+2x_2^2-x_1-8x_2$$

$$\text{s.t.}\begin{cases}-2x_1-3x_2\geqslant -5\\ x_1\geqslant 0,x_2\geqslant 0\end{cases}$$

解：首先确定有关数据：

$$\begin{cases}\boldsymbol{H}=\begin{bmatrix}2 & -1\\ -1 & 4\end{bmatrix}\\ \boldsymbol{c}=\begin{bmatrix}-1\\ -10\end{bmatrix}\\ \boldsymbol{Ae}=[\,]\\ \boldsymbol{be}=[\,]\\ \boldsymbol{Ai}=\begin{bmatrix}-3 & -2\\ 1 & 0\\ 0 & 1\end{bmatrix}\\ \boldsymbol{bi}=\begin{bmatrix}-6\\ 0\\ 0\end{bmatrix}\end{cases}$$

根据以上数据，编写 MATLAB 代码如下：

```
clear all
clc
H = [2 -1; -1 4];
c = [ -1 -8]';
Ae = [ ];
be = [ ];
Ai = [ -2 -3;1 0;0 1];
bi = [ -5 0 0]';
x0 = [0 0]';
[x, lambda, exitflag, output] = qpact(H, c, Ae, be, Ai, bi, x0)

function [x, lambda] = qsubp(H, c, Ae, be)
ginvH = pinv(H);
[m, n] = size(Ae);
if m > 0
    rb = Ae * ginvH * c + be;
    lambda = pinv(Ae * ginvH * Ae') * rb;
    x = ginvH * (Ae' * lambda - c);
else
    x = - ginvH * c;
    lambda = 0;
end
```

```
function [x,lamk,exitflag,output] = qpact(H,c,Ae,be,Ai,bi,x0)
%初始化
epsilon = 1.0e-9;err = 1.0e-6;
k = 0;x = x0;n = length(x);kmax = 1.0e3;
ne = length(be);ni = length(bi);lamk = zeros(ne + ni,1);
index = ones(ni,1);
for i = 1:ni
    if Ai(i,:) * x > bi(i) + epsilon
        index(i) = 0;
    end
end
%算法主程序
while k <= kmax
%求解子问题
    Aee = [];
    if ne > 0
        Aee = Ae;
    end
    for j = 1:ni
        if index(j)> 0
            Aee = [Aee;Ai(j,:)];
        end
    end
    gk = H * x + c;
    [m1,n1] = size(Aee);
    [dk,lamk] = qsubp(H,gk,Aee,zeros(m1,1));
    if norm(dk)<= err
        y = 0.0;
        if length(lamk)> ne
            [y,jk] = min(lamk(ne + 1:length(lamk)));
        end
        if y >= 0
            exitflag = 0;
        else
            exitflag = 1;
            for i = 1:ni
                if index(i)&&(ne + sum(index(1:i))) == jk
                    index(i) = 0;
                    break;
                end
            end
        end
        k = k + 1;
    else
        exitflag = 1;
        %求步长
        alpha = 1.0;tm = 1.0;
        for i = 1:ni
            if (index(i) == 0)&&(Ai(i,:) * dk < 0)
                tm1 = (bi(i) - Ai(i,:) * x)/(Ai(i,:) * dk);
```

```
                    if tm1 < tm
                        tm = tm1;
                        ti = i;
                    end
                end
            end
            alpha = min(alpha,tm);
            x = x + alpha * dk;
            %修正有效集
            if tm < 1
                index(ti) = 1;
            end
        end
        if exitflag == 0
            break;
        end
        k = k + 1;
    end
    output.fval = 0.5 * x' * H * x + c' * x;
    output.iter = k;
```

运行后，得到结果如下：

```
x =
    0.347826086956522
    1.434782608695652
lambda =
    0.869565217391304
exitflag =
    0
output =
    fval: -8.086956521739131
    iter: 7
```

本章小结

二次规划（QP）是指目标函数为决策变量 x 的二次函数，而约束函数是线性函数的非线性规划。二次规划问题是最简单的一类非线性约束优化问题，并且某些非线性规划可以转化为求解一系列二次规划问题，因此二次规划的求解方法也是求解非线性规划的基础之一。

本章重点介绍了二次规划的基本概念，并对其包含的拉格朗日法和起作用集算法两种算法重点讲解，并举例说明 MATLAB 在其中的应用。

第11章 多目标函数的优化方法

多目标最优化是在一定约束条件下，使得多个目标都能达到最优。在现实生活中，很多问题都要求多个目标最好，或者是妥协最好，例如买车要便宜，又要省油，还要快。但是一般来说，多个目标同时达到最优的情况是不存在的。

本章重点介绍四种多目标函数优化方法，包括理想点法、线性加权和法、最大最小法和目标规划法。

学习目标：

(1) 了解目标函数优化方法基本概念；

(2) 掌握 MATLAB 在目标函数优化方法中的应用。

11.1 概述

目标优化问题一般地是指通过一定的优化算法获得目标函数的最优化解。当优化的目标函数为一个时称之为单目标优化(single-objective optimization problem, SOP)。当优化的目标函数有两个或两个以上时称之为多目标优化(multi-objective optimization problem, MOP)。不同于单目标优化的解为有限解，多目标优化的解通常是一组均衡解。

一般来说，多目标决策问题有两类。一类是多目标规划问题，其对象是在管理决策过程中求解使多个目标都达到满意结果的最优方案；另一类是多目标优选问题，其对象是在管理决策过程中根据多个目标或多个准则衡量和得出各种备选方案的优先等级与排序。

多目标优化首先要解决的一个问题是解的存在性问题，这种问题涉及很深的数学理论。其次要解决怎么来求解的问题。

多目标优化的数学模型为

$$\min f_i(x_1, x_2, \cdots, x_n) \quad i = 1, 2, \cdots, s$$

$$\text{s.t. } g_i(x_1, x_2, \cdots, x_n) = 0 \quad i = 1, 2, \cdots, m$$

多目标决策由于考虑的目标多，有些目标之间又彼此有矛盾，使多目标问题成为一个复杂而困难的问题。但由于客观实际的需要，多目标决策问题越来越受到重视，因而出现了许多解决此决策问题的

方法。

一般来说，其基本途径是，把求解多目标问题转化为求解单目标问题。其主要步骤是，先转化为单目标问题，然后利用单目标模型的方法，求出单目标模型的最优解，以此作为多目标问题的解。

化多目标问题为单目标问题的方法大致可分为两类。一类是转化为一个单目标问题，另一类是转化为多个单目标问题，关键是如何转化。

多目标优化算法归结起来有传统优化算法和智能优化算法两大类。

传统优化算法包括加权法、约束法和线性规划法等，实质上就是将多目标函数转化为单目标函数，通过采用单目标优化的方法达到对多目标函数的求解。

智能优化算法包括进化算法、粒子群算法等。

一般来说，多目标优化问题并不存在一个最优解，所有可能的解都称为非劣解，也称为 Pareto 解。传统优化技术一般每次能得到 Pareto 解集中的一个，而用智能算法来求解，可以得到更多的 Pareto 解，这些解构成了一个最优解集，称为 Pareto 最优解。由那些任一个目标函数值的提高都必须以牺牲其他目标函数值为代价的解组成的集合，称为 Pareto 最优域，简称 Pareto 集。

Pareto 最优解非劣解集是指由这样一些解组成的集合：与集合之外的任何解相比它们至少有一个目标函数比集合之外的解好。

求解多目标优化问题最有名的就是 NSGA-II，即多目标遗传算法，但其对解的选择过程可以用在其他优化算法上，例如粒子群，蜂群等。这里简单介绍一下 NSGA-II 的选择算法，主要包含三个部分：

1. 快速非支配排序

要先讲一下支配的概念，对于解 x_1 和 x_2，如果 x_1 对应的所有目标函数都不比 x_2 大（最小问题），且存在一个目标值比 x_2 小，则 x_2 被 x_1 支配。

2. 个体拥挤距离

为了使计算结果在目标空间比较均匀的分布，维持种群多样性，对每个个体计算拥挤距离，选择拥挤距离大的个体。

3. 精英策略选择

精英策略就是保留父代中的优良个体直接进入子代，防止获得的 Pareto 最优解丢失。将第 t 次产生的子代种群和父代种群合并，对合并后的新种群进行非支配排序，然后按照非支配顺序添加到规模为 N 的种群中作为新的父代。

【例 11-1】 求解多目标优化问题

$$\min x^2 \text{ 和 } \min y$$

$$\text{s.t.} \begin{cases} x \leqslant 1 \\ y \geqslant 1 \end{cases}$$

解：从题意中，容易看出，当 $x=0, y=1$ 时，$\min x^2=0$，$\min y=1$。

【例 11-2】 求解多目标优化问题

$$\min f_1(x_1,x_2) = x_1^4 - 5x_1^2 + x_1x_2 + x_2^4 - x_1^2x_2^2$$
$$\min f_2(x_1,x_2) = x_2^4 - x_1^2x_2^2 + x_1^4 + x_1x_2$$
$$\text{s.t.}\begin{cases} -10 \leqslant x_1 \leqslant 10 \\ -10 \leqslant x_2 \leqslant 10 \end{cases}$$

解：首先编写适应值函数 MATLAB 代码如下：

```
function y = f(x)
y(1) = x(1)^4 - 5 * x(1)^2 + x(1) * x(2) + x(2)^4 - x(1)^2 * x(2)^2;
y(2) = x(2)^4 - x(1)^2 * x(2)^2 + x(1)^4 + x(1) * x(2);
```

然后编写求解函数的 MATLAB 代码如下：

```
clear all
clc
fitnessfcn = @f;                    %适应度函数句柄
nvars = 2;                          %变量个数
lb = [ - 10, - 10];                 %下限
ub = [10,10];                       %上限
A = [];b = [];                      %线性不等式约束
Aeq = [];beq = [];                  %线性等式约束
options = gaoptimset('paretoFraction',0.3,'populationsize',...
100,'generations',200,'stallGenLimit',200,'TolFun',1e - 100,'PlotFcns',@gaplotpareto);
[x,fval] = gamultiobj(fitnessfcn,nvars,A,b,Aeq,beq,lb,ub,options)
```

运行程序得到结果如下：

```
Optimization terminated: maximum number of generations exceeded.

x =
   - 0.706730728558836   0.707018045871014
   - 1.951298513728312   1.492779268703866
   - 0.706730728558836   0.707018045871014
   - 1.919078952019131   1.448416110877622
   - 1.768326783462118   1.359283702823248
   - 1.719671669273083   1.240051585047004
   - 1.869118655612149   1.446231023976156
   - 0.892947392414678   0.776450046980439
   - 1.698120840667354   1.355699729260078
   - 1.681425781337095   1.453702659092865
   - 1.488202190717962   1.320716654547650
   - 1.540461308666609   1.289199949159803
   - 1.935250323277039   1.492762708050167
   - 1.380830136085085   1.153096986760025
   - 1.590292629103597   1.326499118564008
   - 1.175698956279529   1.042798205507930
   - 1.904423513728312   1.430279268703866
```

```
  -1.349532423452700   1.006381405725945
  -1.569295282584101   1.334634226616160
  -1.424564060318782   1.255475157662208
  -1.511184763784169   1.240526997599210
  -1.603233573790681   1.138515346105065
  -0.934862722752869   0.837949713204630
  -1.648269528577731   1.291297594036288
  -1.460211205657926   1.280138629236182
  -1.037643512374561   0.845338279764144
  -1.332216678365707   1.049775532250681
  -1.951298513728312   1.492779268703866
  -1.096072392414678   0.901450046980439
  -1.128738198596407   1.063904781484655

fval =
  -2.747341340460560  -0.249999727014046
 -10.972149208956303   8.065680239435295
  -2.747341340460560  -0.249999727014046
 -10.955548864192176   7.458771256222059
 -10.624306460783806   5.010591604763587
 -10.356255619903056   4.430097630599301
 -10.898371014905258   6.569651728881576
  -4.161576494161674  -0.174801266060807
 -10.326883205386842   4.091188742157155
 -10.095975427685199   4.039987863040095
  -8.954731554694893   2.118997247593810
  -9.401521239175091   2.463583978319115
 -10.968427549307735   7.757541519411691
  -8.257493290450409   1.275966033153355
  -9.712569244946945   2.932583985959207
  -6.547309251264204   0.364030927719665
 -10.938630051050943   7.195514547155510
  -7.966211204701748   1.139977605048833
  -9.556855490372683   2.756582929330890
  -8.531310802539529   1.615603007220148
  -9.224006152421660   2.194390799045416
  -9.721918874672518   3.129870585975676
  -4.510025965547547  -0.140184413583005
 -10.081133888341821   3.502828305847451
  -8.792671301272568   1.868412524372300
  -5.360149131271775   0.023371162593302
  -7.864029989726648   1.009976400852145
 -10.972149208956303   8.065680239435295
  -5.867539959565374   0.139333487502809
  -6.108822413883265   0.261427190970049
```

11.2 理想点法

多目标线性规划有着两个或两个以上的目标函数，且目标函数和约束条件全是线性函数，其数学模型表示为

$$\max\begin{cases}\boldsymbol{Z}_1 = c_{11}x_1 + c_{12}x_2 + \cdots + c_{1n}x_n \\ \boldsymbol{Z}_2 = c_{21}x_1 + c_{22}x_2 + \cdots + c_{2n}x_n \\ \quad\vdots \\ \boldsymbol{Z}_r = c_{r1}x_1 + c_{r2}x_2 + \cdots + c_{rn}x_n\end{cases}$$

约束条件为

$$\begin{cases}a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leqslant b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leqslant b_2 \\ \quad\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leqslant b_m \\ x_1, x_2, \cdots, x_n \geqslant 0\end{cases}$$

若数学模型式中只有一个 $z_i = c_{i1}x_1 + c_{i2}x_2 + \cdots + c_{in}x_n$，则该问题为典型的单目标线性规划。记为

$$\boldsymbol{A} = (a_{ij})_{m\times n}$$
$$\boldsymbol{C} = (c_{ij})_{r\times n}$$
$$\boldsymbol{b} = (b_1, b_2, \cdots, b_m)^{\mathrm{T}}$$
$$\boldsymbol{x} = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$$
$$\boldsymbol{Z} = (Z_1, Z_2, \cdots, Z_r)^{\mathrm{T}}$$

则上述多目标线性规划可用矩阵形式表示为

$$\max \boldsymbol{Z} = \boldsymbol{C}\boldsymbol{x}$$

约束条件为

$$\begin{cases}\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b} \\ \boldsymbol{x} \geqslant 0\end{cases}$$

在式 $\max \boldsymbol{Z} = \boldsymbol{C}\boldsymbol{x}$ 中，先求解 r 个单目标问题：$\min\limits_{x\in D} Z_j(x), j = 1, 2, \cdots r$，设其最优值为 Z_j^*，称 $\boldsymbol{Z}^* = (Z_1^*, Z_2^*, \cdots Z_r^*)$ 为值域中的一个理想点，因为一般很难达到。

因此，在期望的某种度量之下，寻求距离 $\boldsymbol{Z}^*$ 最近的 $\boldsymbol{Z}$ 作为近似值。一种最直接的方法是最短距离理想点法，构造评价函数

$$\varphi(Z) = \sqrt{\sum_{i=1}^{r} [Z_i - Z_i^*]^2}$$

然后极小化 $\varphi[Z(x)]$，即求解

$$\min_{x\in D} \varphi[Z(x)] = \sqrt{\sum_{i=1}^{r} [Z_i(x) - Z_i^*]^2}$$

并将它的最优解 $\boldsymbol{x}^*$ 作为 $\max \boldsymbol{Z} = \boldsymbol{C}\boldsymbol{x}$ 在这种意义下的最优解。

【例 11-3】 利用理想点法求解

$$\max f_1(x) = -3x_1 + 2x_2$$
$$\max f_2(x) = 4x_1 + 3x_2$$
$$\text{s. t.} \begin{cases} 2x_1 + 3x_2 \leqslant 18 \\ 2x_1 + x_2 \leqslant 10 \\ x_1, x_2 \geqslant 0 \end{cases}$$

解： 编写 MATLAB 代码，首先分别对单目标求解，然后求如下模型的最优解：

$$\min_{x \in D} \varphi[f(x)] = \sqrt{[f_1(x) - 12]^2 + [f_2(x) - 24]^2}$$
$$\text{s. t.} \begin{cases} 2x_1 + 3x_2 \leqslant 18 \\ 2x_1 + x_2 \leqslant 10 \\ x_1, x_2 \geqslant 0 \end{cases}$$

MATLAB 程序如下：

```
clear all
clc
% 求解 f1(x)
f = [3; - 2];
A = [2,3;2,1];
b = [18;10];
lb = [0;0];
[x1,fval1] = linprog(f,A,b,[],[],lb)
% 求解 f2(x)
f = [ - 4; - 3];
A = [2,3;2,1];
b = [18;10];
lb = [0;0];
[x2,fval2] = linprog(f,A,b,[],[],lb)
% 理想点为(fval1,fval2)
% 继续求解模型最优解
A = [2,3;2,1];
b = [18;10];
x0 = [1;1];
lb = [0;0];
% 最优解
x = fmincon('(( - 3 * x(1) + 2 * x(2) - 12)^2 + (4 * x(1) + 3 * x(2) - 24)^2)^(1/2)',x0,A,b,[],
[],lb,[])
% 对应目标值
f1 = - 3 * x(1) + x(2)
f2 = 4 * x(1) + 3 * x(2)
```

运行得到结果如下：

```
Optimization terminated.

x1 =
```

```
    0.000000002973388
    5.999999996232754
fval1 =
 -11.999999983545344
Optimization terminated.
x2 =
    3.000000000107086
    3.999999999768454
fval2 =
 -23.999999999733703

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the optimality tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.

<stopping criteria details>
x =
    0.526829876328185
    5.648779520964729
f1 =
    4.068289891980173
f2 =
   19.053658068206929
```

即最优解为0.526829876328185、5.648779520964729，对应的目标值分别为4.068289891980173和19.053658068206929。

11.3 线性加权和法

线性加权和法(linear weighted sum method)是一种评价函数的方法，是按各目标的重要性赋予它相应的权系数，然后对其线性组合进行寻优的求解多目标规划问题的方法。

在具有多个指标的问题中，人们总希望对那些相对重要的指标给予较大的权系数，因而将多目标向量问题转化为所有目标的加权求和的标量问题，基于这个现实，构造评价函数如下：

$$\min_{x\in D} Z(x) = \sum_{i=1}^{r} \omega_i Z_i(x)$$

将它的最优解 x^* 作为 $\max \boldsymbol{Z}=\boldsymbol{Cx}$ 在线性加权和意义下的最优解。

其中，ω_i 为加权因子，选取的方法很多，有专家打分法、容限法和加权因子分解法等。

【例 11-4】 对以下数学模型：

$$\max f_1(x) = -3x_1 + 2x_2$$
$$\max f_2(x) = 4x_1 + 3x_2$$

$$\text{s.t.}\begin{cases}2x_1+3x_2\leqslant 18\\2x_1+x_2\leqslant 10\\x_1,x_2\geqslant 0\end{cases}$$

进行线性加权和法求解，其中权系数分别取 $\omega_1=0.7,\omega_2=0.7$。

解：求模型的最优解，首先构造评价函数如下：

$$\min\{0.7\times(3x_1-2x_2)+0.7\times(-4x_1-3x_2)\}$$

$$\text{s.t}\begin{cases}2x_1+3x_2\leqslant 18\\2x_1+x_2\leqslant 10\\x_1,x_2\geqslant 0\end{cases}$$

编写 MATLAB 程序如下：

```
clear all
clc
% 求解 f1(x)
f = [3; -2];
A = [2,3;2,1];
b = [18;10];
lb = [0;0];
[x1,fval1] = linprog(f,A,b,[],[],lb)
% 求解 f2(x)
f = [ -4; -3];
A = [2,3;2,1];
b = [18;10];
lb = [0;0];
[x2,fval2] = linprog(f,A,b,[],[],lb)
% 理想点为(fval1,fval2)
% 继续求解模型最优解
f = [ -0.7  -3.5];
A = [2,3;2,1];
b = [18;10];
lb = [0;0];
% 最优解
x = linprog(f,A,b,[],[],lb)
% 对应目标值
f1 = -3 * x(1) + x(2)
f2 = 4 * x(1) + 3 * x(2)
```

运行程序结果如下：

```
Optimization terminated.

x1 =
    0.000000002973388
    5.999999996232754
fval1 =
-11.999999983545344
```

```
Optimization terminated.
x2 =
   3.000000000107086
   3.999999999768454
fval2 =
 -23.999999999733703

Optimization terminated.
x =
   0.000000006077045
   5.999999993949981
f1 =
   5.999999975718845
f2 =
  18.000000006158125
```

即最优解为 0.000000006077045、5.999999993949981，对应的目标值分别为 5.999999975718845 和 18.000000006158125。

11.4 最大最小法

最大最小法，也叫机会损失最小值决策法，是一种根据机会成本进行决策的方法，它以各方案机会损失大小来判断方案的优劣。

在决策的时候，采取保守策略是稳妥的，即在最坏的情况下，寻求最好的结果，按照此想法，可以构造评价函数如下：

$$\varphi(Z) = \max_{1\leqslant i\leqslant r} Z_i$$

然后求解

$$\min_{x\in D}\varphi[Z(x)] = \min_{x\in D}\max_{1\leqslant i\leqslant r} Z_i(x)$$

并将它的最优解 $\boldsymbol{x}^*$ 作为 $\max\boldsymbol{Z}=\boldsymbol{Cx}$ 在最大最小意义下的最优解。

【例 11-5】 用最大最小法求解以下数学模型：

$$\max f_1(x) = 5x_1 - 2x_2$$
$$\max f_2(x) = -4x_1 - 5x_2$$
$$\text{s.t}\begin{cases}2x_1 + 3x_2 \leqslant 15\\ 2x_1 + x_2 \leqslant 10\\ x_1, x_2 \geqslant 0\end{cases}$$

解：编写 MATLAB 程序如下：

```
clear all
clc
x0 = [1;1];
A = [2,3;2,1];
b = [15;10];
```

```
lb = zeros(2,1);
[x,fval] = fminimax('myfun11_5',x0,A,b,[],[],lb,[])

function f = myfun11_5(x)
f(1) = 5 * x(1) - 2 * x(2);
f(2) = - 4 * x(1) - 5 * x(2);
```

运行程序得到结果如下：

```
Local minimum possible. Constraints satisfied.

fminimax stopped because the size of the current search direction is less than
twice the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.

<stopping criteria details>
x =
   0.000000000000000
   5.000000000000000
fval =
   -10   -25
```

即最优解为0、5，对应的目标值分别为－10和－25。

11.5 目标规划法

目标规划(goal programming)是线性规划的一种特殊应用，能够处理单个主目标与多个目标并存，以及多个主目标与多个次目标并存的问题。由美国学者查纳斯(A. Charnes)和库伯(W. W. Cooper)在1961年首次提出。

目标规划是以线性规划为基础而发展起来的，但在运用中，由于要求不同，有不同于线性规划之处：

(1) 目标规划中的目标不是单一目标而是多目标，既有主要目标又有次要目标。根据主要目标建立部门分目标，构成目标网，形成整个目标体系。制定目标时应注意衡量各个次要目标的权重，各次要目标必须在主要目标完成之后才能给予考虑。

(2) 线性规划只寻求目标函数的最优值，即最大值或最小值。而目标规划，由于是多目标，其目标函数不是寻求最大值或最小值，而是寻求这些目标与预计成果的最小差距，差距越小，目标实现的可能性越大。目标规划中有超出目标和未达目标两种差距。一般以 Y^+ 代表超出目标的差距，Y^- 代表未达目标的差距。

Y^+ 和 Y^- 两者之一必为零，或两者均为零。当目标与预计成果一致时，两者均为零，即没有差距。

目标规划可用一般线性规划求解，也可用备解法求解，还可用单体法求解，或者先用线性规划或备解法求解后，再用单体法验证有无错误。目标规划有时还可以用对偶原理进行运算，依一般规则，将原始问题转换为对偶问题，以减少单体法运算步骤。

在企业中，目标规划的用途极为广泛，如确定利润目标，确定各种投资的收益率，确定产品品种和数量，确定对原材料、外购件、半成品、在制品等数量的控制目标等。

目标规划数学模型为

$$\underset{x\in D}{\mathrm{Appr}}Z(x)\rightarrow \boldsymbol{Z}^0$$

并把原多目标线性规划 $\max \boldsymbol{Z}=\boldsymbol{Cx},\min\limits_{x\in D}Z(x)$称为和目标规划$\underset{x\in D}{\mathrm{Appr}}Z(x)\rightarrow \boldsymbol{Z}^0$ 相对应的多目标线性规划。

为了用数量来描述$\underset{x\in D}{\mathrm{Appr}}Z(x)\rightarrow \boldsymbol{Z}^0$，在目标空间 E^r 中引进点 $Z(x)$与 $\boldsymbol{Z}^0$ 之间的某种“距离”，即

$$D[Z(x),\boldsymbol{Z}^0]=\left[\sum_{i=1}^{r}\lambda_i(Z_i(x)-Z_i^*)^2\right]^{1/2}$$

由此$\underset{x\in D}{\mathrm{Appr}}Z(x)\rightarrow \boldsymbol{Z}^0$ 便可以用单目标$\min\limits_{x\in D}D[Z(x),\boldsymbol{Z}^0]$来描述了。

【例 11-6】 用目标规划法求解以下数学模型：

$$\max f_1(x)=5x_1-3x_2$$

$$\max f_2(x)=-4x_1-7x_2$$

$$\text{s. t}\begin{cases}2x_1+3x_2\leqslant 18\\2x_1+x_2\leqslant 10\\x_1,x_2\geqslant 0\end{cases}$$

解：编写 MATLAB 程序如下：

```
clear all
clc
goal = [18,10];
weight = [18,10];
x0 = [1,1];
A = [2,3;2,1];
b = [18,10];
lb = zeros(2,1);
[x,fval] = fgoalattain('myfun11_6',x0,goal,weight,A,b,[],[],lb,[])

function f = myfun11_6(x)
f(1) = 5 * x(1) - 3 * x(2);
f(2) = - 4 * x(1) - 7 * x(2);
```

运行程序得到结果如下：

```
fminimax stopped because the size of the current search direction is less than
twice the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.

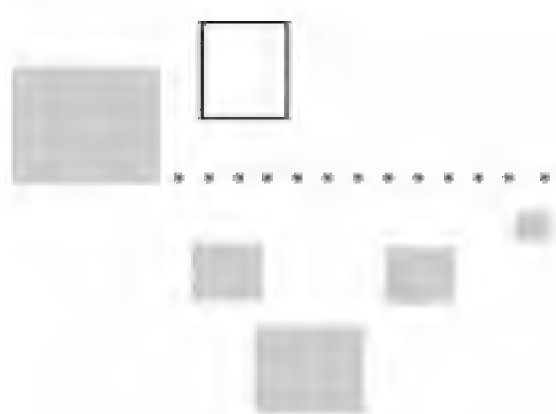
<stopping criteria details>
x =
   - 0.000000000000000   6.000000000000000
fval =
    - 18   - 42
```

即最优解为 0、6，对应的目标值为 −18 和 −42。

本章小结

在现实生活中，往往会有多个决策的目标。例如，对企业产品的生产管理，既希望达到高利润，又希望优质和低消耗，还希望减少对环境的污染等，这是一个多目标决策的问题。又如选购一个好的计算机系统，似乎只有一个目标，但由于要从多方面去反映，要用多个不同的准则来衡量。例如，性能要好，维护要容易，费用要省。这些准则自然构成了多个目标，故也是一个多目标决策问题。

本章首先介绍了多目标函数优化方法的基础知识，随后重点介绍了理想点法、线性加权和法、最大最小法和目标规划法 4 种算法，并举例说明 MATLAB 在多目标函数优化方法中的应用。



第三部分
MATLAB智能优化算法

第12章 免疫优化算法及其实现

免疫算法通过类似于生物免疫系统的机能，构造具有动态性和自适应性的信息防御体系，来抵制外部无用和有害信息的侵入，从而保证接受信息的有效性与无害性。

本章主要介绍免疫算法的基本概念、定义和原理等，并对其运用MATLAB解决最短路径问题做了详细介绍。

学习目标：

(1) 了解免疫算法的基本概念和定义；

(2) 掌握人工免疫系统算法的应用；

(3) 掌握免疫遗传算法的应用；

(4) 熟悉MATLAB在免疫算法求解中的应用。

12.1 基本概念

免疫算法基于生物免疫系统基本机制，模仿了人体的免疫系统。人工免疫系统作为人工智能领域的重要分支，同神经网络及遗传算法一样也是智能信息处理的重要手段，已经受到广泛关注。

基于这一思想，将免疫概念及其理论应用于遗传算法，在保留原算法优良特性的前提下，力图有选择、有目的地利用待求问题中的一些特征信息或知识来抑制其优化过程中出现的退化现象，这种算法称为免疫算法(immune algorithm，IA)。

在生物免疫学的基础上发现，生物免疫系统的运行机制与遗传算法的求解是很类似的。在抵抗抗原时，相关细胞增殖分化进而产生大量抗体。倘若将我们所求的目标函数及约束条件当作抗原，问题的解当作抗体，那么遗传算法求解的过程实际上就是生物免疫系统抵御抗原的过程。

因为免疫系统具有辨识记忆的特点所以可以更快识别个体群体。而我们所说的基于疫苗接种的免疫遗传算法就是将遗传算法映射到生物免疫系统中，结合工程运算得到的一种更高级的优化算法。面对求解问题时，相当于面对各种抗原，可以提前注射“疫苗”来抑制退化问题，从而保持优胜劣汰的特点，使算法一直优化下去，即达到免疫的

目的。

一般的免疫算法可分为三种情况：

(1) 基于免疫系统中的其他特殊机制抽象出的算法，例如克隆选择算法。

(2) 与遗传算法等其他计算智能融合产生的新算法，例如免疫遗传算法。

(3) 模仿免疫系统抗体与抗原识别，结合抗体产生过程而抽象出来的免疫算法。

免疫算法流程图如图 12-1 所示。

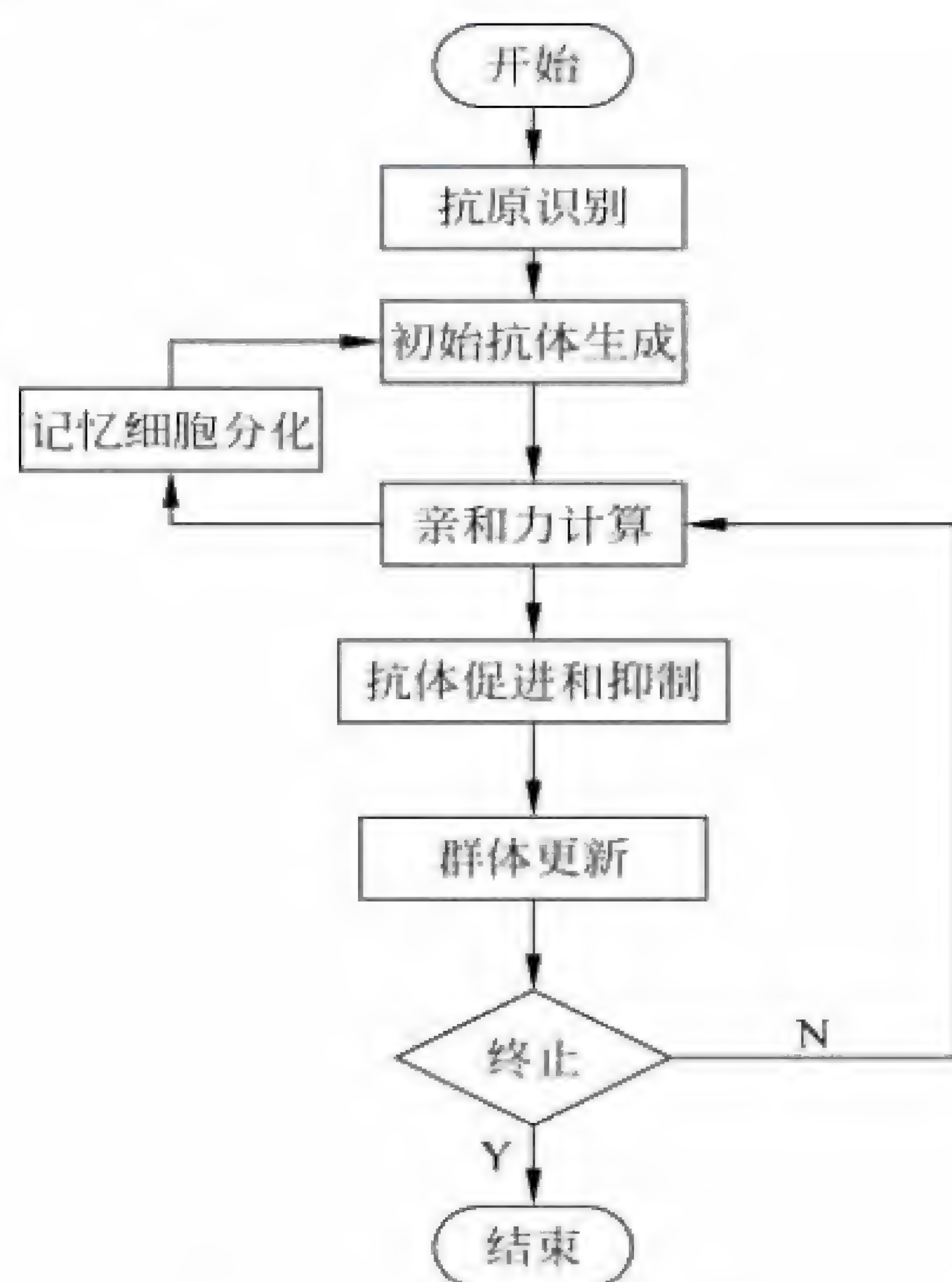


图 12-1　免疫算法流程图

其主要步骤如下：

(1) 抗原识别：输入目标函数和各种约束条件作为免疫算法的抗原，并读取记忆库文件。若问题在文件中有所保留(保留的意思是，该问题以前曾计算过，并在记忆库文件中储存过相关的信息)，则初始化记忆库。

(2) 产生初始解：初始解的产生来源有两种：根据上一步对抗原的识别，如问题在记忆库中有所保留，则提取记忆库，不足部分随机生成；若记忆库为空，则全部随机生成。

(3) 适应度评价(或计算亲和力)：解规模中的各个抗体，按给定的适应度评价函数计算各自适应度。

(4) 记忆单元的更新：将适应度(或期望率)高的个体加入到记忆库中，这保证了对优良解的保留，使其能够延续到后代中。

(5) 基于解的选择：选入适应度(期望率)较高的个体，记其产生后代。所以适应度较低的个体将受到抑制。

(6) 产生新抗体：通过交叉、变异和逆转等算子作用，选入的父代将产生新一代抗体。

(7) 终止条件：条件满足，则终止；不满足，跳转到第(3)步。

其中，免疫算法中最复杂的计算是亲和力计算。由于产生于确定克隆类型的抗体分子独特型是一样的，抗原与抗体的亲和力也是抗体与抗体的亲和力的测量。

一般计算亲和力的公式如下：

$$(A_g)_k = \frac{1}{1 + t_k}$$

其中，t_k是抗原和抗体 k 的结合强度。

一般免疫算法计算结合强度 t_k的数学工具主要如下：

(1) Euclidean 距离，即

$$D = \sqrt{\sum_{i=1}^{L} (x_i - y_i)^2}$$

(2) 海明距离，即

$$D = \sum_{i=1}^{L} \delta \begin{cases} \delta = 1, x_i \neq y_i \\ \delta = 0, \text{其他} \end{cases}$$

(3) Manhattan 距离，即

$$D = \sqrt{\sum_{i=1}^{L} |x_i - y_i|}$$

目前一般免疫算法中抗体抗原，即解和问题的编码方式主要有二进制编码、实数编码和字符编码三种。其中，二进制编码因简单而得到广泛使用。编码后亲和力的计算一般是比较抗体抗原字符串之间的异同，根据上述亲和力计算方法计算。

12.2 人工免疫系统

ARTIS(artificial immune system)是 Hofmeyr 提出的一种分布式人工免疫系统模型，它具有多样性、分布性、错误耐受、动态学习、自适应性和自我监测等特性，可应用于各种工程领域。

ARTIS 的免疫细胞生命周期理论对基于免疫的反垃圾邮件技术具有积极的启迪作用。

ARTIS 模型是一个分布式系统，它由一系列模拟淋巴结的节点构成，每个节点由多个检测器组成。各个节点都可以独立完成免疫功能。模型涉及的免疫机制包括识别、抗体多样性、调节、自体耐受和协同刺激等。

在 ARTIS 中，用固定长度的二进制串构成的有限集合 U 来表示蛋白质链。U 可以分为两个子集：N 表示非自体，S 表示自体，满足

$$U = N \cup S, \quad 且\ N \cap S = \Phi$$

目前常用的两种免疫算法是阴性选择算法和克隆选择算法，其使用格式如下：

1. 克隆选择算法

```
Begin
随机生成一个属性串(免疫细胞)的群体
While 收敛标准没有满足 do
```

```
Begin
While not 所有抗原搜索完毕 do;                           * /初始化 * /
Begin
选择那些与抗原具有更高亲和力的细胞;                       * /选择 * /
生成免疫细胞的副本:越高亲和力的细胞拥有更多的副本;       * /再生 * /
根据它们的亲和力进行变异:亲和力越高,变异越小;           * /遗传变异 * /
end
end
end
```

2. 阴性选择算法

```
procedure
begin
随机生成大量的候选检测器(即免疫细胞)            /* 初始化 */
while 一个给定大小的检测器集合还没有被产生 do   /* 耐受 */
begin
计算出每一个自体元素和一个候选检测器之间的亲和力;
if 这个候选的检测器识别出了自体集合中的任何一个元素
then 这个检测器就要被消除掉;
else 把这个检测器放入检测器集合里面;            /* 该检测器成熟 */
利用经过耐受的检测器集合,检测系统以找出变种;
end
end
```

下面重点介绍克隆选择算法的 MATLAB 应用。

克隆选择原理最先由 Jerne 提出,后由 Burnet 给予完整阐述。其大致内容为当淋巴细胞实现对抗原的识别后,B 细胞被激活并增殖复制产生 B 细胞克隆,随后克隆细胞经历变异过程,产生对抗原具有特异性的抗体。

克隆选择理论描述了获得性免疫的基本特性,并且声明只有成功识别抗原的免疫细胞才得以增殖。经历变异后的免疫细胞分化为效应细胞和记忆细胞两种。

【例 12-1】 用克隆选择算法求解如下模型的最优解:

$$f(x) = x + 5\sin(8x) + 7\cos(3x)$$
$$\text{s.t.} \quad -10 \leqslant x \leqslant 10$$

解:使用 MATLAB 编写代码完成免疫算法的克隆选择如下:

1. 初始化函数

```
function A = InitializeFun(m,n)
A = 2. * rand(m,n) - 1;
A = hardlim(A);
end
```

2. 解码函数

```
function X = DecodeFun(A,xmin,xmax)
A = fliplr(A); % 左右翻转矩阵 A
SA = size(A);
AX = 0:1:21;
AX = ones(SA(1),1) * AX;
SX = sum((A. * 2.^AX)');
X = xmin + (xmax - xmin) * SX./4194303;
End
```

3. 克隆函数

```
%克隆算子
function [T,AAS] = ReproduceFun(mn,cfactor,m,Affinity,A,T);
if mn == 1
    CS = m;
    T = ones(m,1) * A(Affinity(end),:);
else
    for i = 1:mn
        %每个抗体的克隆数与它和抗原的亲和度成正比%
        CS(i) = round(cfactor * m);
        %计算每个抗体的克隆数目 CS(i)
        AAS(i) = sum(CS);
        %每个抗体克隆的最终下标位置
        ONECS = ones(CS(i),1);
        %生成 CS(i)行 1 列单位矩阵 ONECS
        subscript = Affinity(end - i + 1);
        %确定当前要克隆抗体在抗体集 A 中的下标
        AA = A(subscript,:);
        %确定当前要克隆抗体的基因序列集合 AA(1×n)
        T = [T;ONECS * AA];
        %得到零时存放抗体的集合 T
    end
end
```

4. 变异函数

```
%变异算子
function T = Hypermutation(T,n,pMutate,xmax,xmin)
M = rand(size(T,1),n)<= pMutate;
M = T - 2. * (T. * M) + M;
k = round(log(10 * (xmax - xmin)));
k = 1;
T(:,k:n) = M(:,k:n);
End
```

5. 主函数

```
%二维人工免疫优化算法
%m——抗体规模
%n——每个抗体二进制字符串长度
%mn——从抗体集合里选择n个具有较高亲和度的最佳个体进行克隆操作
%A——抗体集合(m×n),抗体的个数为m,每个抗体用n个二进制编码(代表参数)
%T——临时存放克隆群体的集合,克隆规模是抗原亲和度度量的单调递增函数
%FM——每代最大适应度值集合
%FMN——每代平均适应度值集合
%AAS——每个克隆的最终下标位置
%BBS——每代最优克隆的下标位置
%Fit——每代适应度值集合
%tnum——迭代代数
%xymin——自变量下限
%xymax——自变量上限
%pMutate——高频变异概率
%cfactor——克隆(复制)因子
%Affinity——亲和度值大小顺序
%%%%%%%%%%%%%%%%%%
clear all
clc
tic;
m=65;
n=22;
mn=60;
xmin=0;
xmax=8;
tnum=100;
pMutate=0.1;
cfactor=0.3;
A=InitializeFun(m,n); %生成抗体集合A,抗体数目为m,每个抗体基因长度为n
F='X+5*sin(X.*8)+7*cos(X.*3)';         %目标函数
FM=[];                                  %存放各代最优值的集合
FMN=[];                                 %存放各代平均值的集合
t=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while t<tnum
    t=t+1;
    X=DecodeFun(A(:,1:22),xmin,xmax);   %将二进制数转换成十进制数
    Fit=eval(F);                        %以X为自变量求函数值并存放到集合Fit中
    if t==1
        figure(1)
        fplot(F,[xmin,xmax]);
        grid on
        hold on
        plot(X,Fit,'k*')
```

```
        title('抗体的初始位置分布图')
        xlabel('x')
        ylabel('f(x)')
    end
    if t == tnum
        figure(2)
        fplot(F,[xmin,xmax]);
        grid on
        hold on
        plot(X,Fit,'r*')
        title('抗体的最终位置分布图')
        xlabel('x')
        ylabel('f(x)')
    end
    %把零时存放抗体的集合清空
    T = [];
    %把第t代的函数值Fit按从小到大的顺序排列并存放到FS中
    [FS,Affinity] = sort(Fit,'ascend');
    %把第t代的函数值的坐标按从小到大的顺序排列并存放到XT中
    XT = X(Affinity(end - mn + 1:end));
    %从FS集合中取后mn个第t代的函数值按原顺序排列并存放到FT中
    FT = FS(end - mn + 1:end);
    %把第t代的最优函数值加到集合FM中
    FM = [FM FT(end)];
    %克隆(复制)操作,选择mn个候选抗体进行克隆,克隆数与亲和度成正比,……AAS是每个候
选抗体克隆后在T中的开始坐标
    [T,AAS] = ReproduceFun(mn,cfactor,m,Affinity,A,T);
    %把以前的抗体保存到临时克隆群体T里
    T = Hypermutation(T,n,pMutate,xmax,xmin);
    %从大到小重新排列要克隆的mn个原始抗体
    AF1 = fliplr(Affinity(end - mn + 1:end));
    %把以前的抗体保存到临时克隆群体T里%从临时抗体集合T中根据亲和度的值选择mn个
    T(AAS,:) = A(AF1,:);
    X = DecodeFun(T(:,1:22),xmin,xmax);
    Fit = eval(F);
    AAS = [0 AAS];
    FMN = [FMN mean(Fit)];
    for i = 1:mn
        %克隆子群中的亲和度最大的抗体被选中
        [OUT(i),BBS(i)] = max(Fit(AAS(i) + 1:AAS(i + 1)));
        BBS(i) = BBS(i) + AAS(i);
    end
    %从大到小重新排列要克隆的mn个原始抗体
    AF2 = fliplr(Affinity(end - mn + 1:end));
    %选择克隆变异后mn个子群中的最好个体保存到A里,其余丢失
    A(AF2,:) = T(BBS,:);
end
disp(sprintf('\n The optimal point is:'));
disp(sprintf('\n x: %2.4f, f(x):%2.4f',XT(end),FM(end)));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(3)
grid on
plot(FM)
title('适应值变化趋势')
xlabel('迭代数')
ylabel('适应值')
hold on
plot(FMN,'r')
hold off
```

运行主程序，得到抗体的初始位置分布图如图 12-2 所示，抗体最终位置分布图如图 12-3 所示。

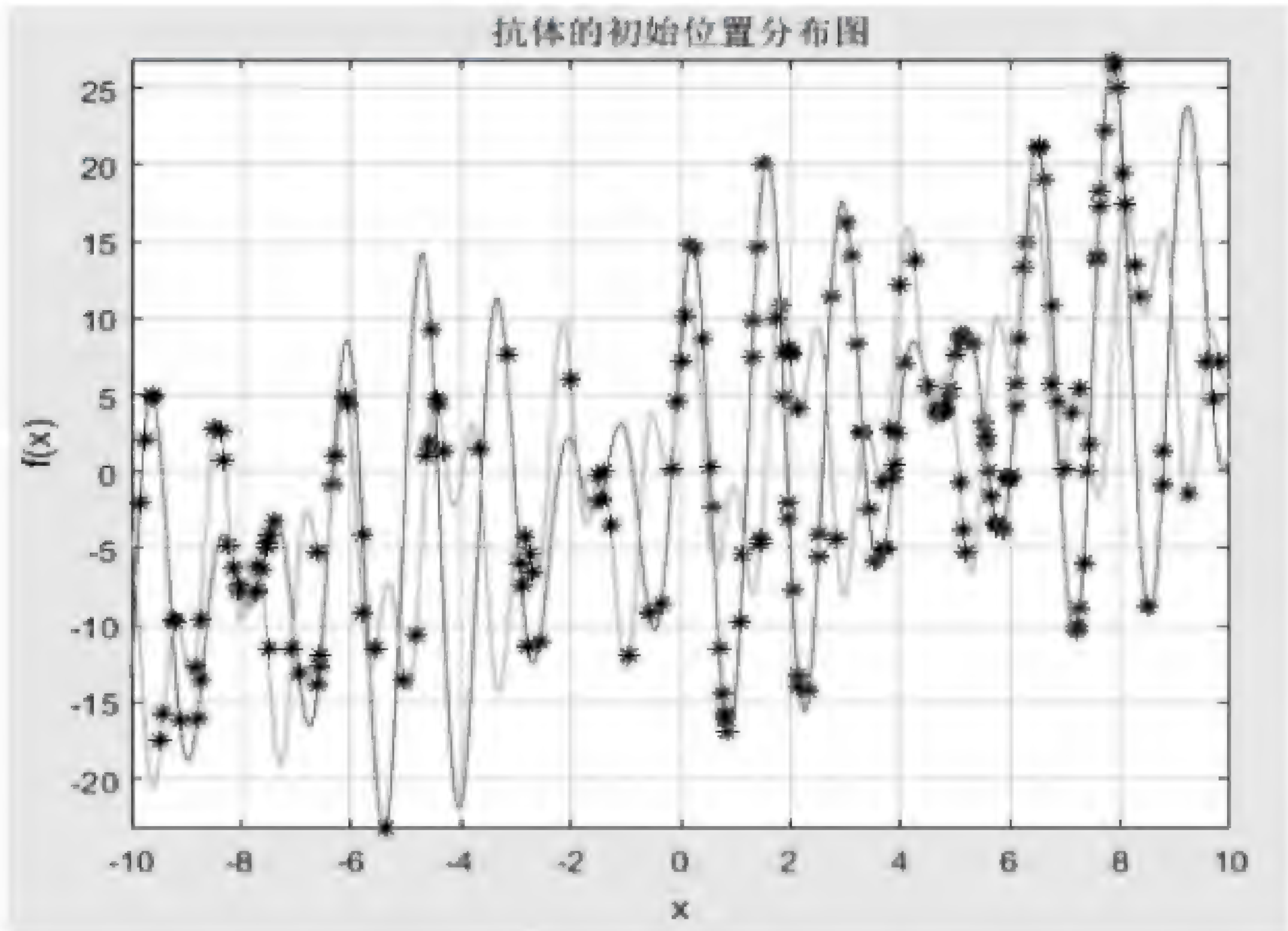


图 12-2　抗体的初始位置分布图

由图 12-2 和图 12-3 中可以看出，子群中的亲和度最大的抗体被克隆，其余抗体被丢弃。

抗体适应值的变化趋势如图 12-4 所示。随着抗体迭代次数的增加，其平均适应值(图 12-4 中虚线 1 所示)和最大适应值(图 12-4 中实线 2)的值趋于稳定。

运行代码后得到结果如下：

```
The optimal point is:
x: 6.4508, f(x):17.4528
```

这说明大约在第 7 代，抗体出现最优适应值为 17.4528。

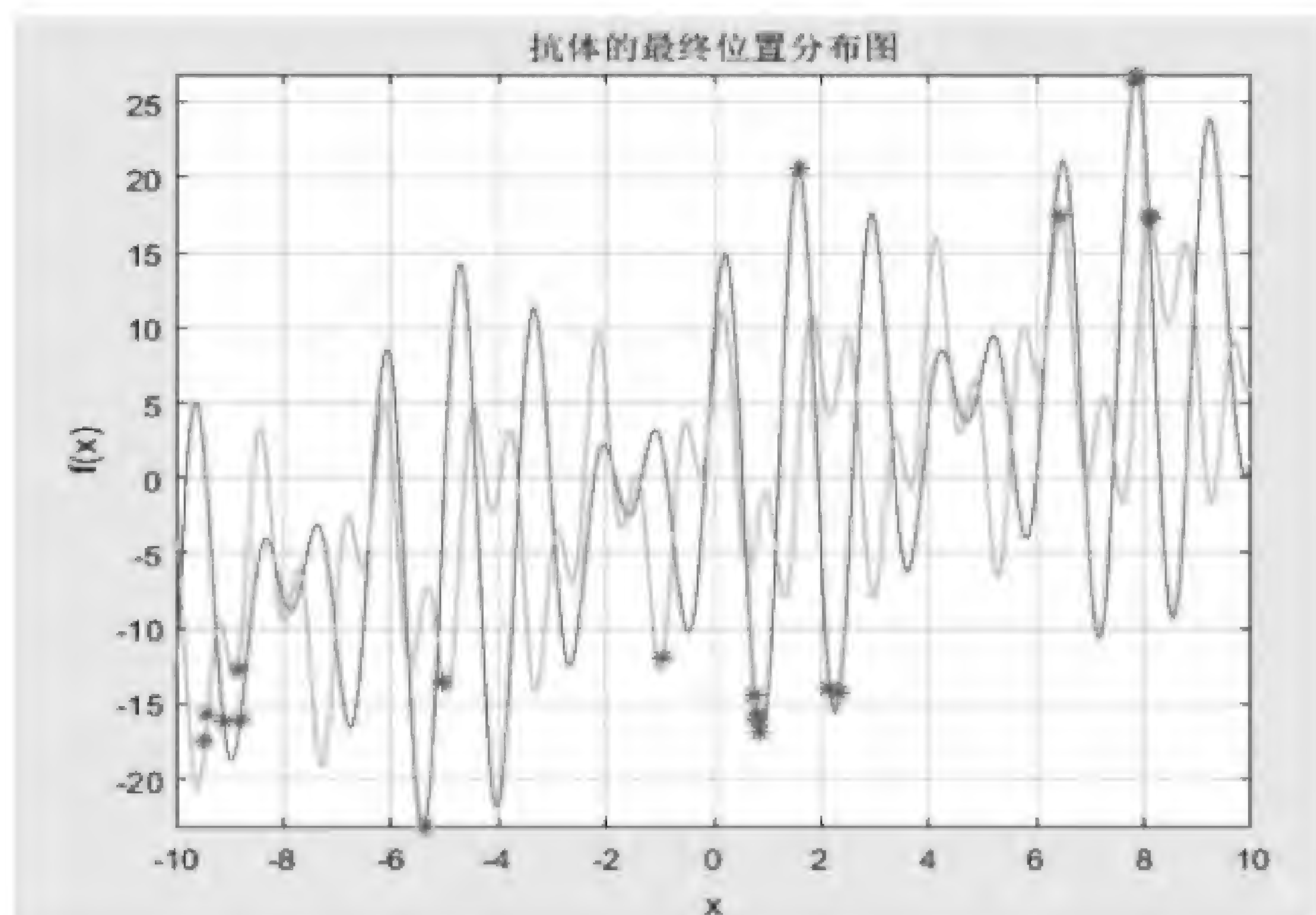


图 12-3 抗体最终位置分布图

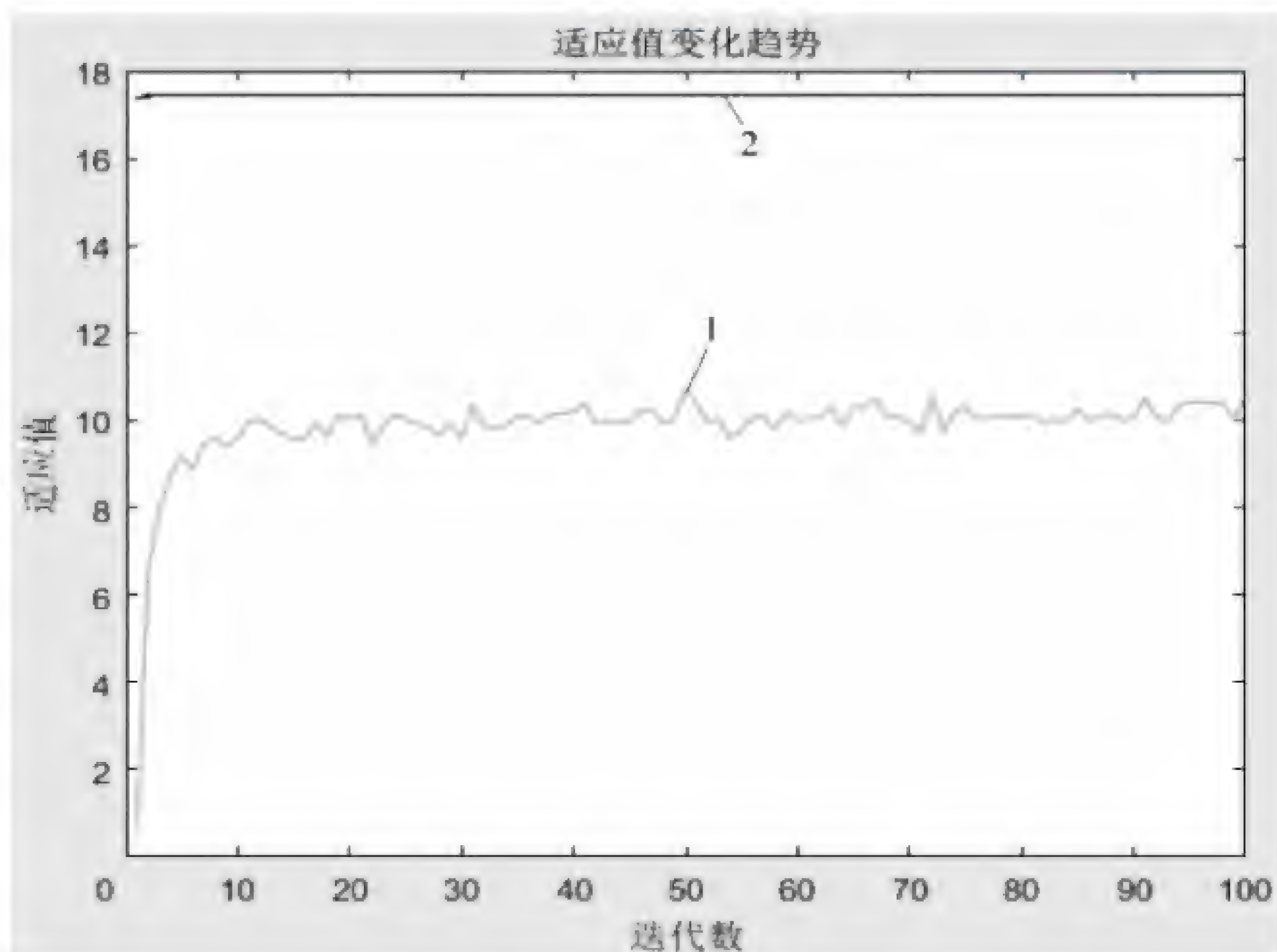


图 12-4 抗体适应值的变化趋势

12.3 免疫遗传算法

免疫遗传算法和遗传算法的结构基本一致，最大的不同之处就在于，在免疫遗传算法中引入了浓度调节机制。进行选择操作时，遗传算法值只利用适应度值指标对个体进行评价；免疫遗传算法的选择策略变为：适应度越高，浓度越小，个体复制的概率越大；适应度越低，浓度越高个体复制的概率越小。

免疫遗传算法的基本思想就是在传统遗传算法的基础上加入一个免疫算子，其目的

是防止种群退化。免疫算子由接种疫苗和免疫选择两个步骤组成免疫遗传算法，可以有效地调节选择压力。因此，免疫算法具有更好的保持群体多样性的能力。

免疫遗传算法的主要步骤如下：

(1) 抗原识别：将目标函数及约束条件当做抗原进行识别，来判定是否曾经解决过该类问题。

(2) 产生初始抗体：对应遗传算法就是解的初始值。经过对抗原的识别，如果曾解决过此类问题，则直接寻找相应记忆细胞，从而产生初始抗体。

(3) 更新记忆单元：选择亲和度高的抗体进行存储记忆。

(4) 抑制和促进抗体：在免疫遗传算法中，由于亲和度高的抗体易受到促进，传进下一代的概率更大，而亲和度低的就会受到抑制，这样很容易导致群体进化单一，导致局部优化。因此需要在算法中插入新的策略，保持群体的多样性。

(5) 遗传操作：经过交叉、变异产生下一代抗体的过程。免疫遗传算法通过考虑抗体亲和度以及群体多样性，选择抗体群体，进行交叉编译从而产生新一代抗体，保证种族向适应度高的方向进化。

用MATLAB实现免疫遗传算法最大优势在于它具有强大的处理矩阵运算的功能。

免疫遗传算法中的标准遗传操作，包括选择、交叉、变异，以及基于生物免疫机制的免疫记忆、多样性保持、自我调节等功能，都是针对抗体(遗传算法称之为个体或染色体)进行的，而抗体可很方便地用向量(即 $1\times n$ 矩阵)表示，因此上述选择、交叉、变异、免疫记忆、多样性保持、自我调节等操作和功能全部由矩阵运算实现的。

【例12-2】 设计一个免疫遗传算法，实现对图12-5所示单阈值图像的分割，并画图比较分割前后图片效果。



图12-5 单阈值图像

解： 图像阈值分割是一种广泛应用的分割技术，利用图像中要提取的目标区域与其背景在灰度特性上的差异，把图像看作具有不同灰度级的两类区域(目标区域和背景区域)的组合，选取一个比较合理的阈值，以确定图像中每个像素点应该属于目标区域还是背景区域，从而产生相应的二值图像。

假设免疫系统群体规模为 N，每个抗体基因长度为 M，采用符号集大小为 S(对二进制编码，$S=2$)，输入变量数为 L(优化问题中指被优化变量个数)，适应度为1，随机产生的新抗体个数 P 为群体规模的40%，进化截止代数为60。

根据图12-5所示，建立MATLAB代码如下：

```
clear all
clc
tic
popsize = 15;
lanti = 10;
maxgen = 60;                                %最大代数
cross_rate = 0.3;                           %交叉速率
mutation_rate = 0.05;                       %变异速率
a0 = 0.6;
zpopsize = 5;
bestf = 0;
nf = 0;
number = 0;
I = imread('fly.png');
q = isrgb(I);                               %判断是否为 RGB 真彩图像
if q == 1
    I = rgb2gray(I);                        %转换 RGB 图像为灰度图像
end
[m,n] = size(I);
p = imhist(I);                              %显示图像数据直方图
p = p';                                     %阵列由列变为行
p = p/(m * n);                              %将 p 的值变换到(0,1)
figure(1)
subplot(1,2,1);
imshow(I);
title('原始图像的灰度图像');
hold on
%%%抗体群体初始化%%%%%%%%%%
pop = 2 * rand(popsize,lanti) - 1;          %pop 的值为(-1,1)之间的随机数矩阵
pop = hardlim(pop);                         %大于等于 0 为 1,小于 0 为 0
%%%%%%免疫操作%%%%%%%%%%%%%%%%%
for gen = 1:maxgen
  [fitness,yuzhi,number] = fitnessty(pop,lanti,I,popsize,m,n,number);
                                                                   %%计算抗体-抗原的亲和度
  if max(fitness)> bestf
    bestf = max(fitness);
    nf = 0;
  for i = 1:popsize
        if fitness(1,i) == bestf            %找出最大适应度在向量 fitness 中的序号
            v = i;
        end
  end
yu = yuzhi(1,v);
  elseif max(fitness) == bestf
    nf = nf + 1;
  end
    if nf >= 20
     break;
     end
A = shontt(pop);                            %计算抗体-抗体的相似度
```

```
f = fit(A,fitness);                          % 计算抗体的聚合适应度
pop = select(pop,f);                         % 进行选择操作
pop = coss(pop,cross_rate,popsize,lanti);    % 交叉
pop = mutation_compute1(pop,mutation_rate,lanti,popsize);                                % 变异
a = shonqt(pop);                             % 计算抗体群体的相似度
if a > a0
    zpop = 2 * rand(zpopsize,lanti) - 1;
    zpop = hardlim(zpop);                    % 随机生成 zpopsize 个新抗体
    pop(popsize + 1:popsize + zpopsize,:) = zpop(:,:);
    [fitness,yuzhi,number] = fitnessty(pop,lanti,I,popsize,m,n,number);
    % 计算抗体 - 抗原的亲和度
    A = shontt(pop);                         % 计算抗体 - 抗体的相似度
    f = fit(A,fitness);                      % 计算抗体的聚合适应度
    pop = select(pop,f);                     % 进行选择操作
end
if gen == maxgen
   [fitness,yuzhi,number] = fitnessty(pop,lanti,I,popsize,m,n,number);
   % 计算抗体 - 抗原的亲和度
end
end
imshow(I);
subplot(1,2,2);
fresult(I,yu);
title('阈值分割后的图像');

% 均匀杂交
function pop = coss(pop,cross_rate,popsize,lanti)
j = 1;
for i = 1:popsize                            % 选择进行抗体交叉的个体
    p = rand;
    if p < cross_rate
        parent(j,:) = pop(i,:);
        a(1,j) = i;
        j = j + 1;
    end
end
j = j - 1;
if rem(j,2)~ = 0
    j = j - 1;
end
for i = 1:2:j
    p = 2 * rand(1,lanti) - 1;               % 随机生成一个模板
    p = hardlim(p);
    for k = 1:lanti
        if p(1,k) == 1
            pop(a(1,i),k) = parent(i + 1,k);
            pop(a(1,i + 1),k) = parent(i,k);
        end
    end
```

```
end

%抗体的聚合适应度函数
function f = fit(A,fitness)
t = 0.8;
[m,m] = size(A);
k = -0.8;
for i = 1:m
    n = 0;
    for j = 1:m
    if A(i,j)>t
        n = n + 1;
    end
    end
    C(1,i) = n/m;                          %计算抗体的浓度
end
f = fitness.*exp(k.*C);                    %抗体的聚合适应度

%适应度计算
function [fitness,b,number] = fitnessty(pop,lanti,I,popsize,m,n,number)
num = m*n;
for i = 1:popsize
     number = number + 1;
        anti = pop(i,:);
        lowsum = 0;                        %低于阈值的灰度值之和
        lownum = 0;                        %低于阈值的像素点的个数
        highsum = 0;                       %高于阈值的灰度值之和
        highnum = 0;                       %高于阈值的像素点的个数
        a = 0;
        for j = 1:lanti
            a = a + anti(1,j)*(2^(j-1));   %加权求和
        end
        b(1,i) = a*255/(2^lanti-1);
        for x = 1:m
            for y = 1:n
                if I(x,y)<b(1,i)
                    lowsum = lowsum + double(I(x,y));
                    lownum = lownum + 1;
                else
                    highsum = highsum + double(I(x,y));
                    highnum = highnum + 1;
                end
            end
        end
        u = (lowsum + highsum)/num;
        if lownum~=0
            u0 = lowsum/lownum;
        else
            u0 = 0;
```

```
        end
        if highnum~ = 0
            u1 = highsum/highnum;
        else
            u1 = 0;
        end
        w0 = lownum/(num);
        w1 = highnum/(num);
        fitness(1,i) = w0 * (u0 - u)^2 + w1 * (u1 - u)^2;
end
end

%根据最佳阈值进行图像分割输出结果
function fresult(I,f,m,n)
[m,n] = size(I);
for i = 1:m
    for j = 1:n
        if I(i,j)<= f
            I(i,j) = 0;
        else
            I(i,j) = 255;
        end
    end
end
imshow(I);

%判断是否为RGB真彩图像
function y = isrgb(x)
wid = sprintf('Images:%s:obsoleteFunction',mfilename);
str1 = sprintf('%s is obsolete and may be removed in the future.',mfilename);
str2 = 'See product release notes for more information.';
warning(wid,'%s\n%s',str1,str2);

y = size(x,3) == 3;
if y
   if isa(x, 'logical')
      y = false;
   elseif isa(x, 'double')
      m = size(x,1);
      n = size(x,2);
      chunk = x(1:min(m,10),1:min(n,10),:);
      y = (min(chunk(:))>= 0 && max(chunk(:))<= 1);
      if y
         y = (min(x(:))>= 0 && max(x(:))<= 1);
      end
   end
end

%变异操作
```

```
function pop = mutation_compute(pop,mutation_rate,lanti,popsize) % 均匀变异
for i = 1:popsize
    s = rand(1,lanti);
    for j = 1:lanti
        if s(1,j)< mutation_rate
            if pop(i,j) == 1
                pop(i,j) = 0;
            else pop(i,j) = 1;
            end
        end
    end
end

 % 选择操作
function v = select(v,fit)
[px,py] = size(v);
for i = 1:px;
pfit(i) = fit(i)./sum(fit);
end
pfit = cumsum(pfit);
if pfit(px)< 1
    pfit(px) = 1;
end
rs = rand(1,10);
for i = 1:10
    ss = 0 ;
    for j = 1:px
        if rs(i)<= pfit(j)
            v(i,:)  =  v(j,:);
            ss = 1;
        end
        if ss == 1
            break;
        end
    end
end

 % 群体相似度函数
function a = shonqt(pop)
[m,n] = size(pop);
h = 0;
for i = 1:n
    s = sum(pop(:,i));
    if s == 0 || s == m
        h = h;
    else
    h = h - s/m * log2(s/m) - (m - s)/m * log2((m - s)/m);
    end
end
```

```
a = 1/(1 + h);

% 抗体相似度计算函数
function A = shontt(pop)
[m,n] = size(pop);
for i = 1:m
    for j = 1:m
        if i == j
            A(i,j) = 1;
        else H(i,j) = 0;
            for k = 1:n
                if pop(i,k)~ = pop(j,k)
                    H(i,j) = H(i,j) + 1;
                end
            end
        H(i,j) = H(i,j)/n;
        A(i,j) = 1/(1 + H(i,j));
        end
    end
end
```

运行以上代码后，可以得到如图 12-6 所示的分割前后图片效果。



(a) 原始图像的灰度图像



(b) 阈值分割后的图像

图 12-6 分割前后图片效果比较图

12.4 免疫算法 MATLAB 应用实例

目前，研究者力求将生命科学中的免疫概念引入到工程实践领域，借助其中的有关知识与理论并将其与已有的一些智能算法有机地结合起来，以建立新的进化理论与算法，来提高算法的整体性能。

本节主要介绍免疫算法的几种典型应用。

12.4.1 最短路径规划

路径规划是指，在具有障碍物的环境中，按照一定的评价标准，寻找一条从起始状态到目标状态的无碰撞路径。最短路径规划问题是图论研究中的一个经典算法问题，其目的是寻找图(由结点和路径组成的)中两结点之间的最短路径。

该问题的具体形式如下：

(1) 确定起点的最短路径问题：已知起始结点，求最短路径的问题。

(2) 确定终点的最短路径问题：与确定起点的问题相反，该问题是已知终结结点，求最短路径的问题。在无向图中该问题与确定起点的问题完全等同，在有向图中该问题等同于把所有路径方向反转的确定起点的问题。

(3) 确定起点终点的最短路径问题：已知起点和终点，求两结点之间的最短路径。

(4) 全局最短路径问题：求图中所有的最短路径。

设 $P(u,v)$是地图中从 u 到 v 的路径，则该路径上的边权之和称为该路径的权，记为 $w(P)$。从 u 到 v 的路径中权最小者 $P*(u,v)$称为 u 到 v 的最短路径。

【例 12-3】 在图 12-7 所示的网络中，求 1 号到 15 号的最短路径。

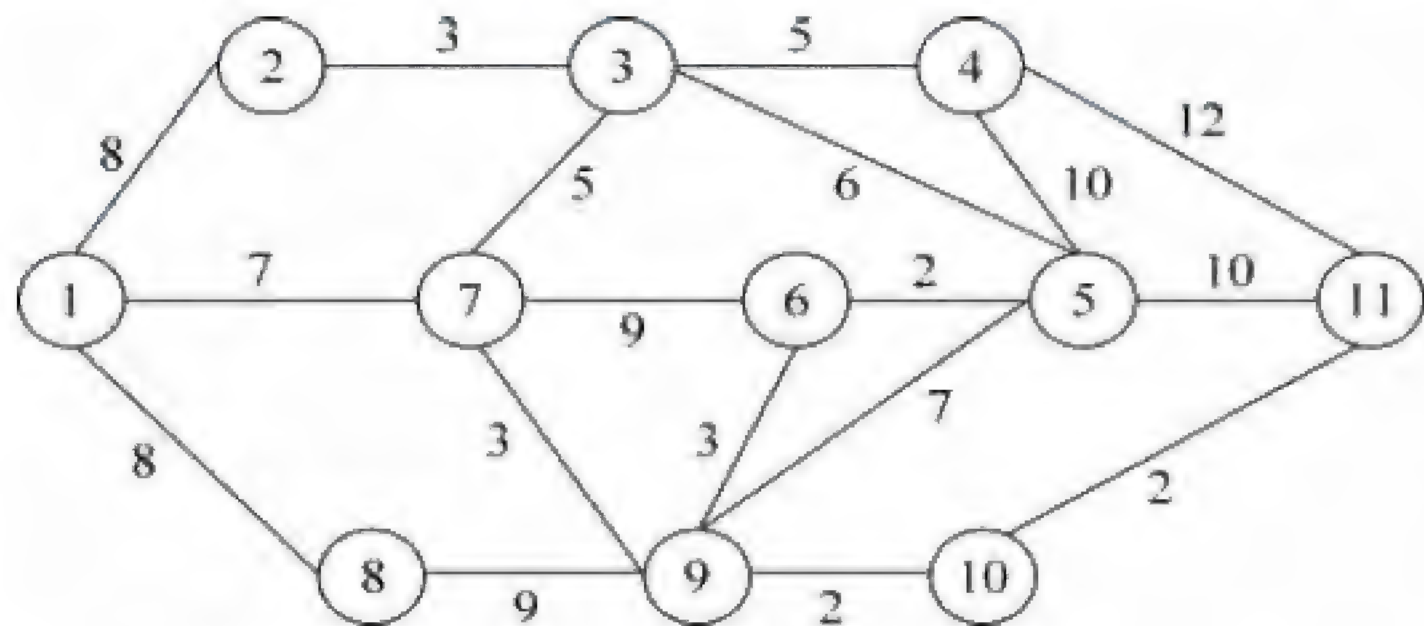


图 12-7 现有交通网络图

解：根据免疫算法原理，在生成初始次优路径时，利用 dist 生成初始次优路径，再根据免疫算法计算全局最优路径。

在 MATLAB 中编写代码如下：

```
function [min,path] = dist(w,start,terminal)
n = size(w,1); label(start) = 0; f(start) = start;
for i = 1:n
   if i~ = start
       label(i) = inf;
end, end
s(1) = start; u = start;
while length(s)<n
   for i = 1:n
      ins = 0;
      for j = 1:length(s)
         if i == s(j)
            ins = 1;
      end, end
      if ins == 0
         v = i;
         if label(v)>(label(u) + w(u,v))
            label(v) = (label(u) + w(u,v)); f(v) = u;
         end
      end
   end
```

```
    v1 = 0;
    k = inf;
    for i = 1:n
            ins = 0;
            for j = 1:length(s)
                if i == s(j)
                    ins = 1;
            end, end
            if ins == 0
                v = i;
                if k > label(v)
                    k = label(v); v1 = v;
    end, end, end
    s(length(s) + 1) = v1;
    u = v1;
end

min = label(terminal); path(1) = terminal;
i = 1;
while path(i)~ = start
        path(i + 1) = f(path(i));
        i = i + 1 ;
end
path(i) = start;
L = length(path);
path = path(L: - 1:1);
```

主函数如下：

```
clear all
clc
edge = [ 2,3,1,3,3,5,4, 4,1,7,6,6,5, 5,11, 1,8,6,9,10,8,9, 9,10;...
                   3,4,2,7,5,3,5,11,7,6,7,5,6,11, 5, 8,1,9,5,11,9,8,10,9;...
                   3,5,8,5,6,6,1,12,7,9,9,2,2,10,10,8,8,3,7, 2, 9,9, 2, 2];
weight = inf * ones(n, n);
for i = 1:n
    weight(i, i) = 0;
end
for i = 1:size(edge,2)
weight(edge(1, i), edge(2, i)) = edge(3, i);
end
[dis, path] = dijkstra(weight, 1,5)
```

运行后得到结果如下：

```
dis =
      5
path =
      1   6   5
```

即表示顶点1到顶点5的最短路径为1→2→3→5，其路径总长度为17。

12.4.2 旅行商问题

免疫算法是在克服遗传算法不足的基础上，提出的一种具有更强鲁棒性和更快收敛速度的搜索算法。它可以很好地解决遗传算法中出现的退化现象，所以在解决复杂的最优问题时具有广泛的应用。

例如，旅行商问题：一个商人从某一城市出发，要遍历所有目标城市，其中每个城市必须而且只须访问一次。其具体过程如下：

1. 个体编码和适应度函数

（1）算法实现中，将TSP问题的目标函数对应于抗原，问题的解对应于抗体。

（2）抗体采用以遍历城市的次序排列进行编码，每一抗体码串形如：$V_1,V_2,\cdots,V_n$，其中V_i表示遍历城市的序号。适应度函数取路径长度T_d倒数：

$$\text{Fitness}(i) = 1/T_d(i)$$

其中，$T_d(i) = \sum_{i=1}^{n-1} d(v_i,v_{i+1}) + d(v_n,v_1)$表示第$i$个抗体所表示的遍历路径长度。

2. 交叉与变异算子

采用单点交叉，其中交叉点的位置随机确定。算法中加入了对遗传个体基因型特征的继承性和对进一步优化所需个体特征的多样性进行评测的环节，在此基础上设计了一种部分路径变异法。

该方法每次选取全长路径的一段，路径子段的起点与终点由评测的结果估算确定。具体操作为采用连续n次的调换方式，其中n的大小由遗传代数K决定。

3. 免疫算子

免疫算子有两种类型全免疫（非特异性免疫）和目标免疫（特异性免疫），其中全免疫即群体中的每个个体在进化算子作用后，对其每一环节都进行一次免疫操作的免疫类型；目标免疫即在进行了进化操作后，经过一定的判断，个体仅在作用点处发生免疫反应的一种类型。

对于旅行商问题，要找到适用于整个抗原（即全局问题求解）的疫苗极为困难，所以我们采用目标免疫。

在求解问题之前先从每个城市点的周围各点中选取一个路径最近的点，以此作为算法执行过程中对该城市点进行目标免疫操作时所注入的疫苗。

每次遗传操作后，随机抽取一些个体注射疫苗，然后进行免疫检测，即对接种了疫苗的个体进行检测：若适应度提高，则继续；反之，若其适应度不如父代，说明在交叉、变异的过程中出现了严重的退化现象，这时该个体将被父代中所对应的个体所取代。

在选择阶段，先计算其被选中的概率，后进行相应的条件判断。

【例12-4】 下面选取8个城市的规模，随机生成城市的坐标。

解：使用免疫算法解决旅行商问题的 MATLAB 代码如下：

```
clear all
clc
N = 8;
%城市的个数
M = N - 1;
%种群的个数
pos = randn(N,2);
%%生成城市的坐标
global D;
%城市距离数据
D = zeros(N,N);
for i = 1:N
    for j = i + 1:N
        dis = (pos(i,1) - pos(j,1)).^2 + (pos(i,2) - pos(j,2)).^2;
        D(i,j) = dis^(0.5);
        D(j,i) = D(i,j);
    end
end

%中间结果保存
global TmpResult;
TmpResult = [];
global TmpResult1;
TmpResult1 = [];

%参数设定
[M, N] = size(D);                            %集群规模
pCharChange = 1;                             %字符换位概率
pStrChange = 0.3;                            %字符串移位概率
pStrReverse = 0.3;                           %字符串逆转概率
pCharReCompose = 0.3;                        %字符重组概率
MaxIterateNum = 100;                         %最大迭代次数

%数据初始化
mPopulation = zeros(N - 1,N);
mRandM = randperm(N - 1);                    %最优路径
mRandM = mRandM + 1;
for rol = 1:N - 1
    mPopulation(rol,:) = randperm(N);        %产生初始抗体
    mPopulation(rol,:) = DisplaceInit(mPopulation(rol,:));     %预处理
end

%迭代
count = 0;
figure(2);
while count < MaxIterateNum
    %产生新抗体
    B = Mutation(mPopulation, [pCharChange pStrChange pStrReverse pCharReCompose]);
    %计算所有抗体的亲和力和所有抗体和最优抗体的排斥力
    mPopulation = SelectAntigen(mPopulation,B);
```

```
    hold on
    plot(count,TmpResult(end),'o');
    drawnow
display(TmpResult(end));
display(TmpResult1(end));
    count = count + 1;
end

hold on
plot(TmpResult,'-r');
title('最佳适应度变化趋势')
xlabel('迭代数')
ylabel('最佳适应度')
%mRandM

function result = CharRecompose(A)
global D;
index = A(1,2:end);
tmp = A(1,1);
result = [tmp];
[m,n] = size(index);
while n>=2
    len = D(tmp,index(1));
    tmpID = 1;
    for s = 2:n
        if len > D(tmp,index(s))
            tmpID = s;
            len = D(tmp,index(s));
        end
    end
    tmp = index(tmpID);
    result = [result,tmp];
    index(:,tmpID) = [];
    [m,n] = size(index);
end
result = [result,index(1)];

%预处理
function result = DisplaceInit(A)
[m,n] = size(A);
tmpCol = 0;
for col = 1:n
    if A(1,col) == 1
        tmpCol = col;
        break;
    end
end
if tmpCol == 0
    result = [];
```

```
else
    result = [A(1,tmpCol:n), A(1,1:(tmpCol - 1))];
end

function result = DisplaceStr(inMatrix, startCol, endCol)
[m,n] = size(inMatrix);
if n <= 1
    result = inMatrix;
    return;
end
switch nargin
    case 1
        startCol = 1;
        endCol = n;
    case 2
        endCol = n;
end
mMatrix1 = inMatrix(:,(startCol + 1):endCol);
result = [mMatrix1, inMatrix(:, startCol)];

function result = InitAntigen(A, B)
[m,n] = size(A);
global D;
Index = 1:n;
result = [B];
tmp = B;
Index(:,B) = [];
for col = 2:n
    [p,q] = size(Index);
    tmplen = D(tmp,Index(1,1));
    tmpID = 1;
    for ss = 1:q
        if D(tmp,Index(1,ss)) < tmplen
            tmpID = ss;
            tmplen = D(tmp,Index(1,ss));
        end
    end
    tmp = Index(1,tmpID);
    result = [result tmp];
    Index(:,tmpID) = [];
End

function result = Mutation(A, P)
[m,n] = size(A);
%字符换位
n1 = round(P(1) * m);
m1 = randperm(m);
cm1 = randperm(n - 1) + 1;
B1 = zeros(n1,n);
```

```
c1 = cm1(n-1);
c2 = cm1(n-2);
for s = 1:n1
    B1(s,:) = A(m1(s),:);
    tmp = B1(s,c1);
    B1(s,c1) = B1(s,c2);
    B1(s,c2) = tmp;
end

%字符串移位
n2 = round(P(2) * m);
m2 = randperm(m);
cm2 = randperm(n-1) + 1;
B2 = zeros(n2,n);
c1 = min([cm2(n-1),cm2(n-2)]);
c2 = max([cm2(n-1),cm2(n-2)]);
for s = 1:n2
    B2(s,:) = A(m2(s),:);
    B2(s,c1:c2) = DisplaceStr(B2(s,:),c1,c2);
end

%字符串逆转
n3 = round(P(3) * m);
m3 = randperm(m);
cm3 = randperm(n-1) + 1;
B3 = zeros(n3,n);
c1 = min([cm3(n-1),cm3(n-2)]);
c2 = max([cm3(n-1),cm3(n-2)]);
for s = 1:n3
    B3(s,:) = A(m3(s),:);
    tmp1 = [[c2:-1:c1]',B3(s,c1:c2)'];
    tmp1 = sortrows(tmp1,1);
    B3(s,c1:c2) = tmp1(:,2)';
end

%字符重组
n4 = round(P(4) * m);
m4 = randperm(m);
cm4 = randperm(n-1) + 1;
B4 = zeros(n4,n);
c1 = min([cm4(n-1),cm4(n-2)]);
c2 = max([cm4(n-1),cm4(n-2)]);
for s = 1:n4
    B4(s,:) = A(m4(s),:);
    B4(s,c1:c2) = CharRecompose(B4(s,c1:c2));
end

%
result = [B1;B2;B3;B4];
```

```
function result = SelectAntigen(A,B)
global D;
[m,n] = size(A);
[p,q] = size(B);
index = [A;B];
rr = zeros((m+p),2);
rr(:,2) = [1:(m+p)]';
for s = 1:(m+p)
    for t = 1:(n-1)
        rr(s,1) = rr(s,1)+D(index(s,t),index(s,t+1));
    end
    rr(s,1) = rr(s,1) + D(index(s,n),index(s,1));
end
rr = sortrows(rr,1);
ss = [];
tmplen = 0;
for s = 1:(m+p)
    if tmplen ~= rr(s,1)
        tmplen = rr(s,1);
        ss = [ss;index(rr(s,2),:)];
    end
end
global TmpResult;
TmpResult = [TmpResult;rr(1,1)];
global TmpResult1;
TmpResult1 = [TmpResult1;rr(end,1)];
result = ss(1:m,:);
```

运行以上代码，得到如图 12-8 所示的最佳适应度变化趋势。

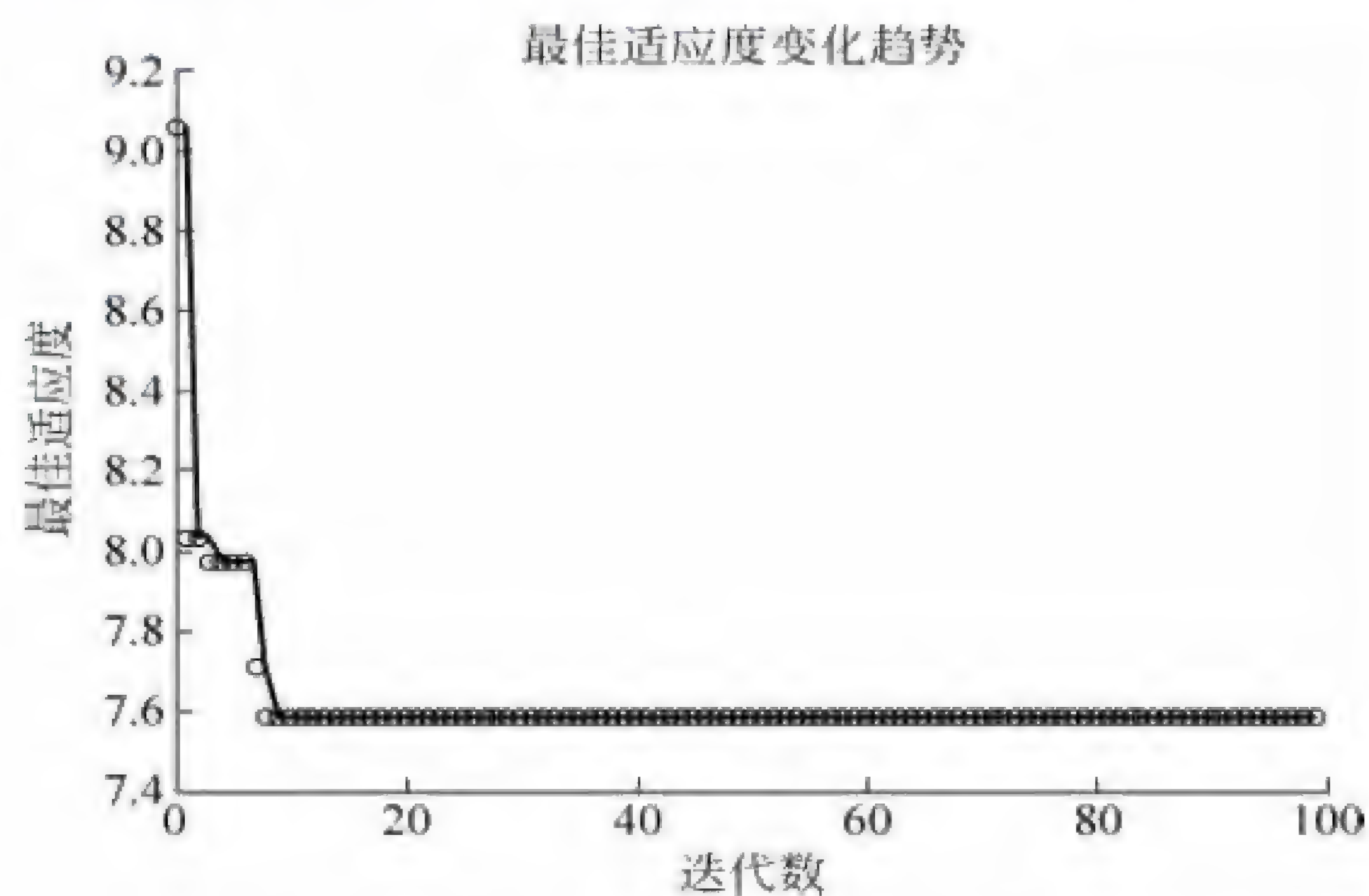


图 12-8　最佳适应度变化趋势

在 MATLAB 命令行窗口输入最优路径变量 mRandM，可以得到

```
mRandM =

     4   2   5   7   8   3   6
```

即 8 个城市的最优路径为 4→2→5→7→8→3→6。

12.4.3 故障检测问题

免疫算法的基础就在于如何计算抗原与抗体、抗体与抗体之间的相似度，因此免疫算法在处理相似性方面有着独特的优势。

基于人工免疫的故障检测和诊断模型如图 12-9 所示。

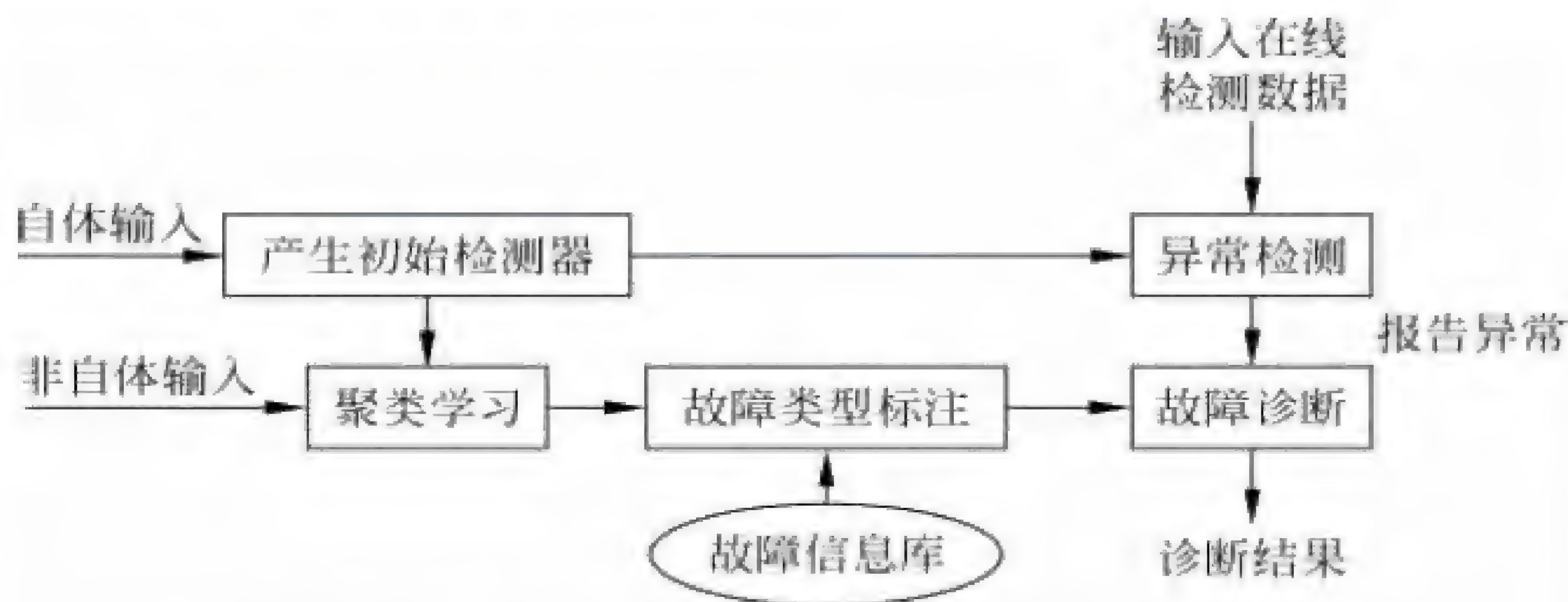


图 12-9 基于人工免疫的故障检测和诊断模型

在此模型中，用一个 N 维特征向量表示系统工作状态的数据。为了减少时间的复杂度，对系统工作状态的检测分为如下两个层次：

(1) 异常检测：负责报告系统的异常工作状态。

(2) 故障诊断：确定故障类型和发生的位置。

描述系统正常工作的自体为第一类抗原，用于产生原始抗体；描述系统工作异常的非自体作为第二类抗原，用于刺激抗体进行变异和克隆进化，使其成熟。

下面采用免疫算法对诊断知识的获取技术进行举例讲解。

【例 12-5】 随机设置一组故障编码和三种故障类型编码，通过免疫算法，求得故障编码属于故障类型编码的概率。

解：根据故障检测模型，编写 MATLAB 代码如下：

```
clear all
clc
global popsize length min max N code;
N = 11;                 % 每个染色体段数(十进制编码位数)
M = 110;                % 进化代数
popsize = 20;           % 设置初始参数,群体大小
length = 10;            % length 为每段基因的二进制编码位数
chromlength = N * length; % 字符串长度(个体长度),染色体的二进制编码长度
pc = 0.7;
 % 设置交叉概率,本例中交叉概率是定值,若想设置变化的交叉概率可用表达式表示……
```

```
%或从写一个交叉概率函数,例如用神经网络训练得到的值作为交叉概率
pm = 0.3;          %设置变异概率,同理也可设置为变化的
bound = { - 100 * ones(popsize,1),zeros(popsize,1)};min = bound{1};max = bound{2};
pop = initpop(popsize,chromlength);
%运行初始化函数,随机产生初始群体
ymax = 500;
K = 1;

%故障类型编码,每一行为一种!code(1,:),正常; code(2,:),50%; code(3,:),100%
code =[ - 0.8180  - 1.6201  - 14.8590  - 17.9706  - 24.0737  - 33.4498  - 43.3949
- 53.3849  - 63.3451  - 73.0295  - 79.6806  - 74.3230;  - 0.7791  - 1.2697
- 14.8682  - 26.2274  - 30.2779  - 39.4852  - 49.4172  - 59.4058  - 69.3676
- 79.0657  - 85.8789  - 81.0905;
- 0.8571  - 1.9871  - 13.4385  - 13.8463  - 20.4918  - 29.9230  - 39.8724
- 49.8629  - 59.8215  - 69.4926  - 75.9868  - 70.6706];
%设置故障数据编码
Unnoralcode = [ - 0.5164  - 5.6743  - 11.8376  - 12.6813  - 20.5298  - 39.9828
- 43.9340  - 49.9246  - 69.8820  - 79.5433  - 65.9248  - 8.9759];

for i = 1:3
%3种故障模式,每种模式应该产生 popsize 种监测器(抗体),每种监测器的长度和故障编码的长
度相同
   for k = 1:M %判断每种模式适应值
       [objvalue] = calobjvalue(pop,i);        %计算目标函数
       fitvalue = calfitvalue(objvalue);
       favg(k) = sum(fitvalue)/popsize;        %计算群体中每个个体的适应度
       newpop = selection(pop,fitvalue);
       objvalue = calobjvalue(newpop,i);       %选择
       newpop = crossover(newpop,pc,k);
       objvalue = calobjvalue(newpop,i);       %交叉
       newpop = mutation(newpop,pm);
       objvalue = calobjvalue(newpop,i);       %变异
       for j = 1:N                             %译码
           temp(:,j) = decodechrom(newpop,1 + (j - 1) * length,length);
           %将 newpop 每行(个体)每列(每段基因)转化成十进制数
           x(:,j) = temp(:,j)/(2 ^ length - 1) * (max(j) - min(j)) + min(j);
           %popsize×N 将二值域中的数转化为变量域的数
       end
       [bestindividual,bestfit] = best(newpop,fitvalue);
       %求出群体中适应值最大的个体及其适应值
       if bestfit < ymax
          ymax = bestfit;
          K = k;
       end
       %y(k) = bestfit;
       if ymax < 10                            %如果最大值小于设定阈值,停止进化
           X{i} = x;
           break
       end
       if k == 1
```

```
            fitvalue_for = fitvalue;
            x_for = x;
        end
        result = resultselect(fitvalue_for,fitvalue,x_for,x);
        fitvalue_for = fitvalue;
        x_for = x;
        pop = newpop;
    end
    X{i} = result;
 % 第 i 类故障的 popsize 个监测器
    distance = 0;
    % 计算 Unnoralcode 属于每一类故障的概率
    for j = 1:N
        distance = distance + (result(:,j) - Unnoralcode(j)).^2; % 将得到 N 个不同的距离
    end
    distance = sqrt(distance);
    D = 0;
    for p = 1:popsize
        if distance(p)< 40           % 预设阈值
            D = D + 1;
        end
    end
    P(i) = D/popsize                 % Unnoralcode 隶属每种故障类型的概率
end

X ;                                  % 结果为(i * popsie)个监测器(抗体)
plot(1:M,favg)
title('个体适应度变化趋势')
xlabel('迭代数')
ylabel('个体适应度')

 %%%%%%% 子函数 %%%%%%%%
 % 求出群体中适应值最大的个体及其适应值
function [bestindividual,bestfit] = best(pop,fitvalue)
global popsize N length;
bestindividual = pop(1,:);
bestfit = fitvalue(1);
for i = 2:popsize
     if fitvalue(i)> bestfit         % 最大的个体
        bestindividual = pop(i,:);
        bestfit = fitvalue(i);
     end
end

 % 计算个体的适应值,目标: 产生可比较的非负数值
function fitvalue = calfitvalue(objvalue)
fitvalue = objvalue;
global popsize;
Cmin = 0;
for i = 1:popsize
```

```
  if objvalue(i) + Cmin > 0              % objvalue 为一列向量
     temp = Cmin + objvalue(i);
   else
     temp = 0;
   end
   fitvalue(i) = temp;                   % 得到一个向量
end
end

%实现目标函数的计算,交叉
function [objvalue] = calobjvalue(pop,i)
global length N min max code;
%默认染色体的二进制长度 length = 10
distance = 0;
for j = 1:N
    temp(:,j) = decodechrom(pop,1 + (j - 1) * length,length);
     %将 pop 每行(个体)每列(每段基因)转化成十进制数
    x(:,j) = temp(:,j)/(2^length - 1) * (max(j) - min(j)) + min(j);
     % popsize×N 将二值域中的数转化为变量域的数
    distance = distance + (x(:,j) - code(i,j)).^2;
     %将得到 popsize 个不同的距离
end
objvalue = sqrt(distance);
%计算目标函数值: 欧氏距离
end

function newpop = crossover(pop,pc,k)
global N length M;
pc = pc - (M - k)/M * 1/20;
A = 1:N * length;
% A = randcross(A,N,length);             %将数组 A 的次序随机打乱(可实现两两随机配对)
for i = 1:length
    n1 = A(i);n2 = i + 10;               %随机选中的要进行交叉操作的两个染色体
    for j = 1:N                          % N 点(段)交叉
           cpoint = length - round(length * pc);    %这两个染色体中随机选择的交叉的位置
           temp1 = pop(n1,(j - 1) * length + cpoint + 1:j * length);temp2 = pop(n2,(j - 1) *
length + cpoint + 1:j * length);
           pop(n1,(j - 1) * length + cpoint + 1:j * length) = temp2;pop(n2,(j - 1) * length +
cpoint + 1:j * length) = temp1;
    end
    newpop = pop;
end
end

%产生 [2^n 2^(n-1) ... 1] 的行向量,然后求和,将二进制转化为十进制
function pop2 = decodebinary(pop)
[px,py] = size(pop);                     %求 pop 行数和例数
for i = 1:py
pop1(:,i) = 2.^(py - 1). * pop(:,i);
```

```
%pop 的每一个行向量(二进制表示)
py = py - 1;
%乘以权重
end
pop2 = sum(pop1,2);
%求 pop1 的每行之和,即得到每行二进制表示变为十进制表示值,实现二进制到十进制的转变
end

%将二进制编码转换成十进制,参数 spoint 表示待解码的二进制串的起始位置
%(对于多个变量而言,如有两个变量,采用 20 为表示,每个变量 10 为,则第一个变量从 1 开始,
另一个变量从 11 开始.本例为 1)
%参数 length 表示所截取的长度
function pop2 = decodechrom(pop,spoint,length)
pop1 = pop(:,spoint:spoint + length - 1);
%将从第"spoint"位开始到第"spoint + length - 1"位(这段码位表示一个参数)取出
pop2 = decodebinary(pop1);
%利用上面函数"decodebinary(pop)"将用二进制表示的个体基因变为十进制数,得到 popsize×1
列向量
end

%置换
function B = hjjsort(A)
N = length(A);t = [0 0];
for i = 1:N
    temp(i,2) = A(i);
    temp(i,1) = i;
end
for i = 1:N - 1 %沉底法将 A 排序
    for j = 2:N + 1 - i
        if temp(j,2)< temp(j - 1,2)
            t = temp(j - 1,:);temp(j - 1,:) = temp(j,:);temp(j,:) = t;
        end
    end
end
for i = 1:N/2 %将排好的 A 逆序
    t = temp(i,2);temp(i,2) = temp(N + 1 - i,2);temp(N + 1 - i,2) = t;
end
for i = 1:N
    A(temp(i,1)) = temp(i,2);
end
B = A;

%编码初始化编码
%initpop.m 函数的功能是实现群体的初始化,popsize 表示群体的大小,chromlength 表示染色体
的长度(二值数的长度)
%长度大小取决于变量的二进制编码的长度
function pop = initpop(popsize,chromlength)
pop = round(rand(popsize,chromlength));
%rand 随机产生每个单元为 {0,1} 行数为 popsize,列数为 chromlength 的矩阵
```

```
%roud 对矩阵的每个单元进行圆整.这样产生随机的初始种群
end

%变异操作
function [newpop] = mutation(pop,pm)
global popsize N length;
for i = 1:popsize
   if(rand < pm)                               %产生一个随机数与变异概率比较
      mpoint = round(rand * N * length);       %个体变异位置
     if mpoint <= 0
        mpoint = 1;
     end
     newpop(i,:) = pop(i,:);
     if newpop(i,mpoint) == 0
        newpop(i,mpoint) = 1;
     else
        newpop(i,mpoint) = 0;
     end
   else
     newpop(i,:) = pop(i,:);
   end
end

function result = resultselect(fitvalue_for,fitvalue,x_for,x);
global popsize;
A = [fitvalue_for;fitvalue];B = [x_for;x];
N = 2 * popsize;
t = 0;
for i = 1:N
    temp1(i) = A(i);
    temp2(i,:) = B(i,:);
end
for i = 1:N - 1                                %沉底法将 A 排序
    for j = 2:N + 1 - i
        if temp1(j)<temp1(j - 1)
            t1 = temp1(j - 1);t2 = temp2(j - 1,:);
            temp1(j - 1) = temp1(j);temp2(j - 1,:) = temp2(j,:);
            temp1(j) = t1;temp2(j,:) = t2;
        end
    end
end
for i = 1:popsize                              %将 A 的低适应值(前一半)的序号取出
    result(i,:) = temp2(i,:);
end

function [newpop] = selection(pop,fitvalue)
global popsize;
fitvalue = hjjsort(fitvalue);
```

```
totalfit = sum(fitvalue);          % 求适应值之和
fitvalue = fitvalue/totalfit;      % 单个个体被选择的概率
fitvalue = cumsum(fitvalue);       % 如 fitvalue = [4 2 5 1],则 cumsum(fitvalue) = [4 6 11 12]
ms = sort(rand(popsize,1));
% 从小到大排列,将"rand(px,1)"产生的一列随机数变成轮盘赌形式的表示方法,由小到大排列
fitin = 1;
% fivalue 是一向量,fitin 代表向量中元素位,即 fitvalue(fitin)代表第 fitin 个个体的单个个体被选择的概率
newin = 1;
while newin <= popsize
      if (ms(newin))< fitvalue(fitin)
% ms(newin)表示的是 ms 列向量中第"newin"位数值,同理 fitvalue(fitin)
          newpop(newin,:) = pop(fitin,:);
% 赋值 ,即将旧种群中的第 fitin 个个体保留到下一代(newpop)
          newin = newin + 1;
      else
          fitin = fitin + 1;
      end
end
```

运行以上代码,得到个体适应度变化趋势如图 12-10 所示。

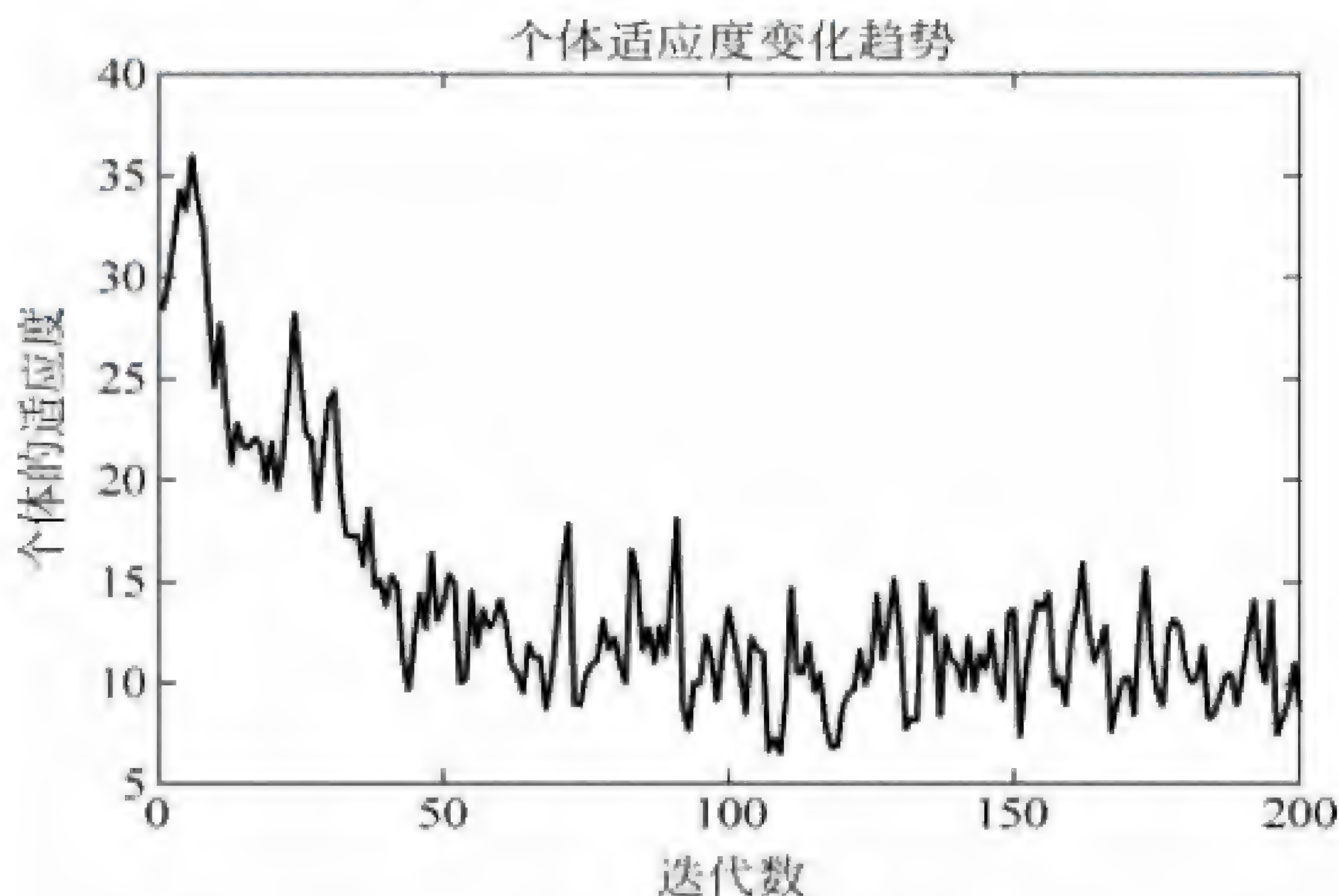


图 12-10 个体适应度变化趋势

设置的故障数据属于三种故障类型的概率 P 值如下:

```
P =

  0.800000000000000   0.050000000000000   1.000000000000000
```

这表示故障数据属于故障一概率为 80%,属于故障二的概率为 5%,属于故障三的概率为 100%。

本章小结

随着研究领域问题的不断深入，常规的确定性算法越来越不能满足人们的需求，包括免疫算法在内的智能算法的应用越来越广泛。

本章首先全面介绍了免疫算法的基本概念，然后分别介绍了人工免疫系统和免疫遗传算法的应用，最后举例说明免疫算法的 MATLAB 实现。

第13章 粒子群优化算法的实现

粒子群优化算法以其实现容易、精度高、收敛快等优点引起了学术界的重视，并且在解决实际问题中展示了其优越性。粒子群算法是一种并行算法。

本章主要介绍了粒子群算法的原理及其在 MATLAB 上的运用。

学习目标：

(1) 了解粒子群优化算法基本概念；

(2) 掌握粒子群优化算法的 MATLAB 实现；

(3) 熟练掌握粒子群权重控制算法的应用；

(4) 熟练掌握混合粒子群算法的应用。

13.1 算法的基本概念

粒子群优化算法属于进化算法的一种，和模拟退火算法相似，它也是从随机解出发，通过迭代寻找最优解，也是通过适应度来评价解的品质，但它比遗传算法规则更为简单，它没有遗传算法的“交叉”(Crossover)和“变异”(Mutation)操作，它通过追随当前搜索到的最优值来寻找全局最优。

粒子群算法，也称粒子群优化算法(particle swarm optimization，PSO)。PSO 从这种模型中得到启示并用于解决优化问题。PSO 中，每个优化问题的潜在解都是搜索空间中的一只鸟，称为粒子。所有的粒子都有一个被优化函数决定的适值(fitness value)，每个粒子还有一个速度决定它们“飞行”的方向和距离，然后粒子们就追随当前的最优粒子在解空间中搜索。

粒子位置的更新方式如图 13-1 所示。

图 13-1 中，x 表示粒子起始位置，v 表示粒子“飞行”的速度，p 表示搜索到的粒子的最优位置。

PSO 初始化为一群随机粒子(随机解)，然后通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个极值来更新自己：第一个就是粒子本身所找到的最优解，这个解称为个体极值；另一个是整个种群目前找到的最优解，这个极值是全局极值。另外也可以不用整个种

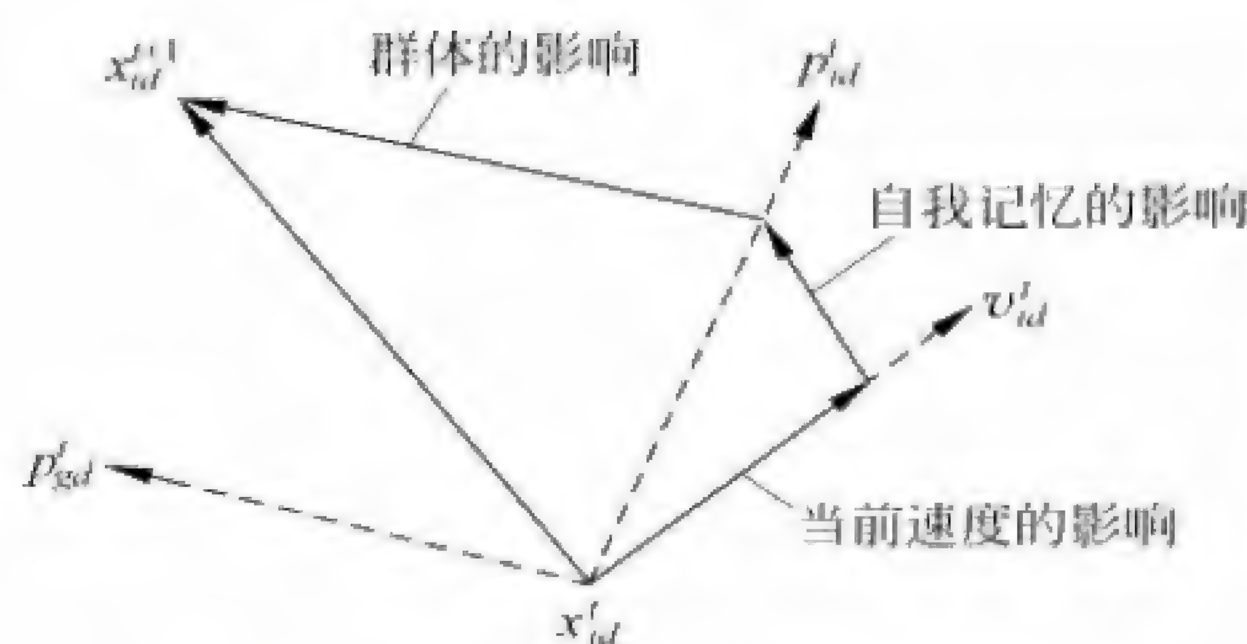


图 13-1　每代粒子位置的更新方式

群而只是用其中一部分作为粒子的邻居，那么在所有邻居中的极值就是局部极值。

假设在一个 D 维的目标搜索空间中，有 N 个粒子组成一个群落，其中第 i 个粒子表示为一个 D 维的向量

$$\boldsymbol{X}_i = (x_{i1}, x_{i2}, \cdots, x_{iD}), \quad i = 1,2,\cdots,N$$

第 i 个粒子的“飞行”速度也是一个 D 维的向量，记为

$$\boldsymbol{V}_i = (v_{i1}, v_{i2}, \cdots, v_{iD}), \quad i = 1,2,3$$

第 i 个粒子迄今为止搜索到的最优位置称为个体极值，记为

$$\boldsymbol{p}_{\text{best}}(p_{i1}, p_{i2}, \cdots, p_{iD}), \quad i = 1,2,\cdots,N$$

整个粒子群迄今为止搜索到的最优位置为全局极值，记为

$$\boldsymbol{g}_{\text{best}} = (p_{g1}, p_{g2}, \cdots, p_{gD})$$

在找到这两个最优值时，粒子根据如下的公式：

$$v_{id} = wv_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id})$$

$$x_{id} = x_{id} + v_{id}$$

来更新自己的速度和位置。其中，c_1 和 c_2 为学习因子，也称加速常数（acceleration constant），r_1 和 r_2 为[0,1]范围内的均匀随机数。

式 $v_{id} = wv_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id})$右边由以下三部分组成：

第一部分为“惯性(inertia)”或“动量(momentum)”部分，反映了粒子的运动“习惯(habit)”，代表粒子有维持自己先前速度的趋势；

第二部分为“认知(cognition)”部分，反映了粒子对自身历史经验的记忆(memory)或回忆(remembrance)，代表粒子有向自身历史最佳位置逼近的趋势；

第三部分为“社会(social)”部分，反映了粒子间协同合作与知识共享的群体历史经验，代表粒子有向群体或邻域历史最佳位置逼近的趋势。

由于粒子群算法具有高效的搜索能力，有利于得到多目标下的最优解；通过代表整个解集种群，按并行方式同时搜索多个非劣解，即搜索到多个 Pareto 最优解。

同时，粒子群算法的通用性比较好，适合处理多种类型的目标函数和约束，并且容易与传统的优化方法结合，从而改进自身的局限性，更高效地解决问题。因此，将粒子群算法应用于解决多目标优化问题上具有很大的优势。

13.2 算法的 MATLAB 实现

基本粒子群算法使用固定长度的二进制符号串来表示群体中的个体，其等位基因是由二值符号集{0,1}所组成的。初始群体中各个个体的基因值可用均匀分布的随机数来生成。

13.2.1 算法的基本程序

其基本粒子群算法描述如下：

```
begin
     Initalize ;                              % 包括初始化粒子群数，粒子初始速度和位置
         [x,xd] = judge(x,pop_size);  % 调用 judge 函数，初始化第一次值

     for num = 2:最大迭代次数
            wk = wmax - num * (wmax - wmin)/max_gen; % 计算惯性权重
            r1 =  ; r2 =                      % 随机产生加速权重
            PSO 算法
       迭代求 vk,xk;
       While 判断 vk 是否满足条件
              再次重新生成加速权重系数 r1;r2
              PSO 算法
              再次迭代求 vk;xk 数值
        end
        调用[x,xd] = judge(x,pop_size);重新计算出目标函数值
        判断并重新生成 pj 数值;
        判断并重新生成 pjd 数值;
        if 迭代前数值>迭代后的数值
            累加迭代次数值
        end
        输出随机数种子、进度、最优迭代次数、每个函数的数值和目标函数的数值
        用 ASCII 保存粒子位移的数值
        用 ASCII 保存粒子速度的数值
     end
```

在 MATLAB 中编程实现的基本粒子群算法基本函数为 PSO，其调用格式如下：

```
[xm,fv] = PSO(fitness,N,c1,c2,w,M,D)
```

其中，fitness 为待优化的目标函数，也称适应度函数。N 是粒子数目，c1 是学习因子 1，c2 是学习因子 2，w 是惯性权重，M 是最大迭代数，D 是自变量的个数，xm 是目标函数取最小值时的自变量，fv 是目标函数的最小值。

使用 MATLAB 实现基本粒子群算法是代码如下：

```
function[xm,fv] = PSO(fitness,N,c1,c2,w,M,D)
%%%%%%%给定初始化条件%%%%%%%%%
%c1 学习因子 1
%c2 学习因子 2
%w 惯性权重
%M 最大迭代次数
%D 搜索空间维数
%N 初始化群体个体数目
%%%%%初始化种群的个体(可以在这里限定位置和速度的范围)%%%%%%%%%
format long;
for i = 1:N
    for j = 1:D
        x(i,j) = randn;             %随机初始化位置
        v(i,j) = randn;             %随机初始化速度
    end
end
%%%%%%先计算各个粒子的适应度,并初始化 Pi 和 Pg%%%%%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                        %pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:)) < fitness(pg)
        pg = x(i,:);
    end
end
%%%%%进入主要循环,按照公式依次迭代,直到满足精度要求%%%%%
for t = 1:M
    for i = 1:N                     %更新速度、位移
        v(i,:) = w*v(i,:) + c1*rand*(y(i,:) - x(i,:)) + c2*rand*(pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:)) < p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)< fitness(pg)
            pg = y(i,:);
        end
    end
    Pbest(t) = fitness(pg);
end
%%%%%%最后给出计算结果
disp('**************************************************')
disp('目标函数取最小值时的自变量:')
xm = pg'
disp('目标函数的最小值为:')
fv = fitness(pg)
disp('**************************************************')
```

将上面的函数保存到MATLAB可搜索路径中，即可调用该函数。再定义不同的目标函数fitness和其他输入量，就可以用粒子群算法求解不同问题。

13.2.2 适应度函数

适应度表示个体 x 对环境的适应程度，分为两类。一类为针对被优化的目标函数的优化型适应度，另一类为针对约束函数的约束性适应度。

粒子适应度是反应粒子当前位置优劣的一个参数。对应某些具有较高适应度的粒子 p_i，在 p_i 所在的局部区域可能存在能够更新全局最优的点 p_x，即 p_x 表示的解要优于全局最优。为了使全局最优能够迅速更新，从而迅速找到 p_x，应该减小粒子 p_i 惯性权重以增强其局部寻优能力；而对于适应度较低的粒子，当前位置较差，所在区域存在优于全局最优解的概率较低，为了跳出当前的区域，应当增大惯性权重，增强全局搜索能力。

其中优化型适应度和约束型适应度分别表示为

$$F_{\text{obj}}(X) = f(x)$$

$$F_i(X) = \begin{cases} 0, & g_i(X) \leqslant 0 \\ g_i(X), & g_i(X) > 0 \end{cases}$$

定义不同约束条件的权重为 ω_i，则总的约束型适应度为 $f_{\text{con}}(X) = \sum_{i=1}^{m} \omega_i F_i(X)$ 。其中，$\sum_{i=1}^{m} \omega_i = 1, 0 \leqslant \omega_i \leqslant 1$ ，这里 ω_i 随机获得，任意选取 k 组 ω_i，获得的 k 个适应值的均值作为最终的总的约束型适应度。

粒子群算法使用的适应度函数有很多种，下面介绍两个经典的适应度函数。

1. GW函数

该函数的MATLAB代码如下：

```
function y = GW(x)
% 输入 x,给出相应的 y 值,在 x = (0,0, …,0)处有全局极小点 0
[row,col] = size(x);
if row > 1
    error('输入的参数错误');
end
y1 = 1/4000 * sum(x.^2);
y2 = 1;
for h = 1:col
    y2 = y2 * cos(x(h)/sqrt(h));
end
y = y1 - y2 + 1;
y = - y;
```

绘制以上函数图形，可以编写MATLAB代码如下：

```
clear all
clc
x = [ - 7:0.05:7];
y = x;
[X,Y] = meshgrid(x,y);
[row,col] = size(X);
for l = 1:col
   for h = 1:row
        z(h,l) = GW([X(h,l),Y(h,l)]);
   end
end
surf(X,Y,z);
shading interp
```

运行后得到 GW 函数图像如图 13-2 所示。

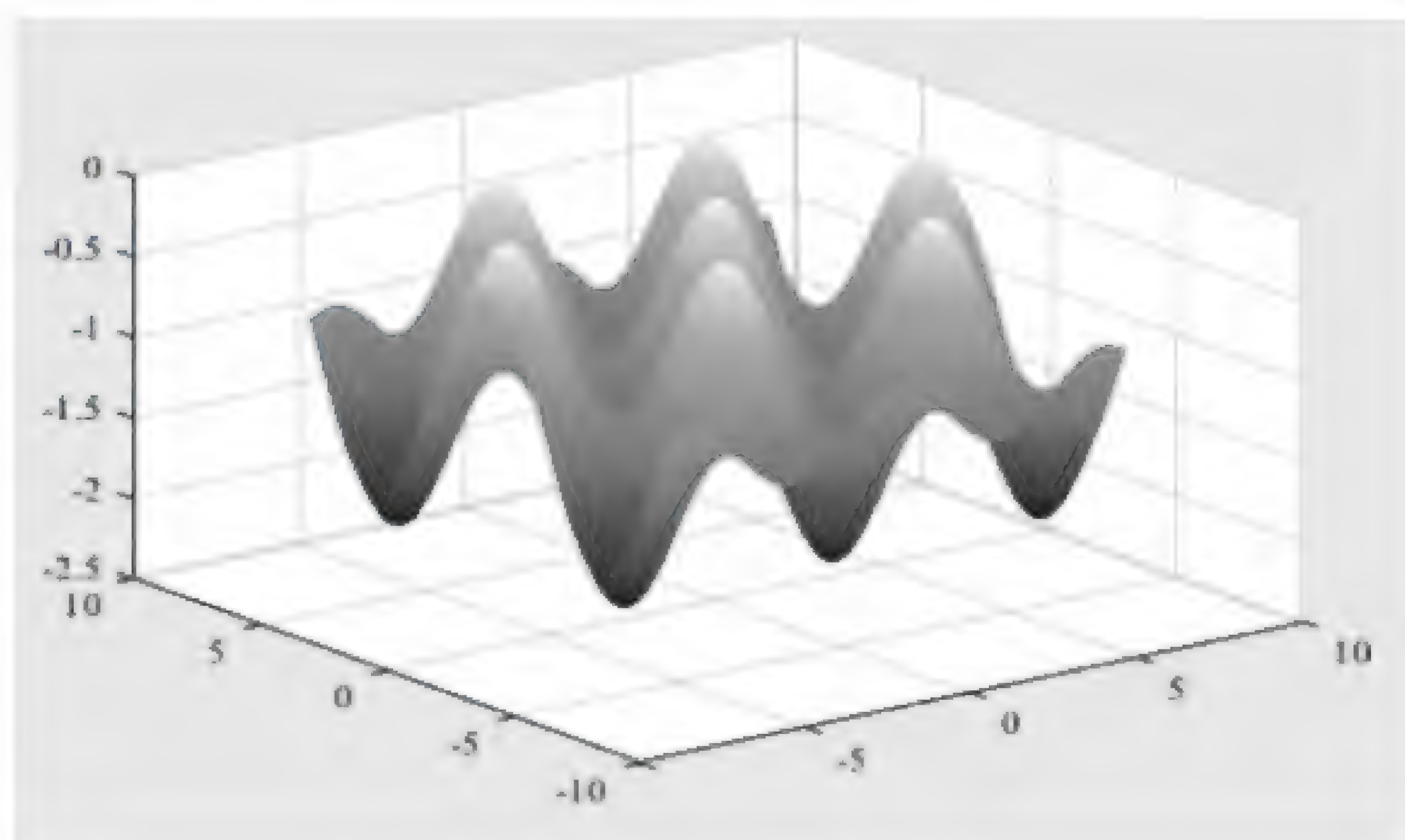


图 13-2 GW 函数图像

2. RA 函数

该函数 MATLAB 代码如下：

```
function y = RA(x)
% 输入 x,并给出相应的 y 值,在 x = (0,0,…,0)处有全局极小点 0
[row,col] = size(x);
if row > 1
    error('输入的参数错误');
end
y = sum(x.^2 - 10 * cos(2 * pi * x) + 10);
y = - y;
```

绘制函数图的 MATLAB 代码如下：

```
clear all
clc
x = [ - 7:0.05:7];
y = x;
[X,Y] = meshgrid(x,y);
[row,col] = size(X);
for l = 1:col
    for h = 1:row
        z(h,l) = Rastrigin([X(h,l),Y(h,l)]);
    end
end
surf(X,Y,z);
shading interp
```

运行上述代码，得到 RA 函数图像如图 13-3 所示。

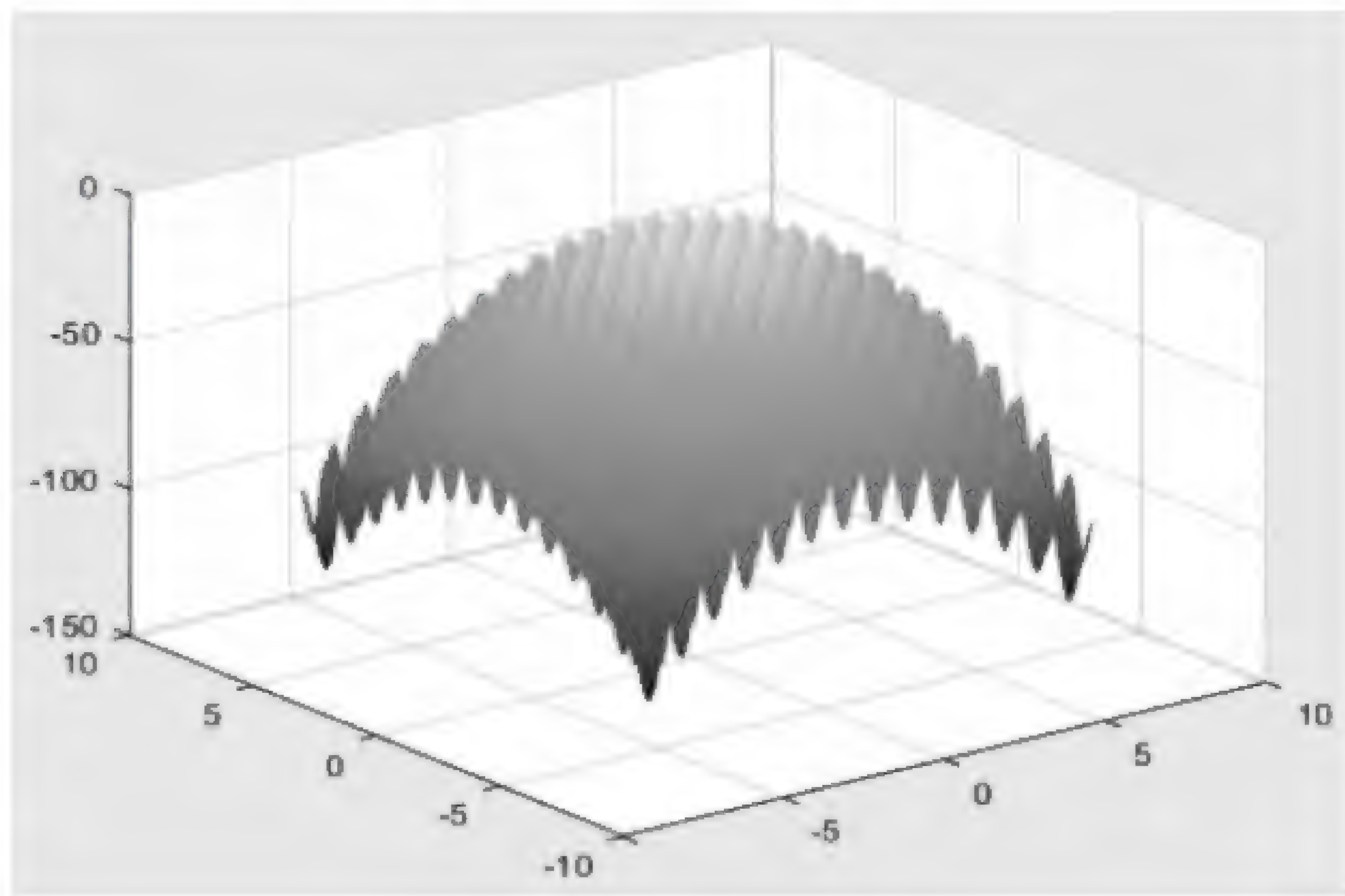


图 13-3　RA 函数图像

【例 13-1】 利用上述介绍的基本粒子群算法求解下列函数的最小值。

$$f(x) = \sum_{i=1}^{10} x_i^2 + 2x_i - 3$$

解：利用 PSO 算法求解最小值，需要首先确认不同迭代步数对结果的影响。设定题中函数的最小点均为 0，粒子群规模为 40，惯性权值为 0.6，学习因子 1 为 1.2，学习因子 2 为 2.2，迭代步数为 100 和 300。

在 MATLAB 中建立目标函数代码如下：

```
function F = fitness(x)
F = 0;
for i = 1:10
    F = F + x(i)^2 + 2 * x(i) - 3;
end
```

编写函数最小值求解代码如下：

```
clear all
clc
x = zeros(1,10);
[xm,fv] = PSO(@fitness,40,1.2,2.2,0.6,100,10);
%取自变量
Xm;
%取函数最小值
Fv;
```

运行后得到自变量和目标函数最小值如下：

```
**************************************************
目标函数取最小值时的自变量：
xm =
   -0.999047610301838
   -1.000167711924916
   -1.002434926707148
   -0.997509720383734
   -0.998668611101255
   -0.997357117816384
   -1.005356615637108
   -1.007432926982811
   -0.996556682139123
   -0.996111538331053

目标函数的最小值为：
fv =
 -39.999867258736643
***************************************************
```

当迭代步数为300，其他参数保持不变，编写代码如下：

```
clear all
clc
x = zeros(1,10);
[xm,fv] = PSO(@fitness,40,1.2,2.2,0.6,300,10);
%取自变量
Xm;
%取函数最小值
Fv;
```

运行后得到结果如下：

```
**************************************************
目标函数取最小值时的自变量：

xm =
```

```
    -0.999167608666782
    -1.000651150731285
    -0.998288518516341
    -0.999842582970857
    -1.001734404212912
    -0.997779983276688
    -1.000494709626516
    -0.998087873234064
    -1.002992895690611
    -0.998721638675077

目标函数的最小值为:
fv =
-39.999973499947501
**********************************************************
```

PSO 算法是一种随机算法,同样的参数也会算出不同的结果,且迭代步数越大,获得解的精度不一定越高。

在粒子群算法中,要想获得精度高的解,关键各个参数之间的合理搭配。

13.2.3 免疫粒子群算法的 MATLAB 应用

基于免疫的混合粒子群算法步骤如下:

(1) 确定学习因子 c_1 和 c_2、粒子(抗体)群体个数 M。

(2) 由 logistic 回归分析映射产生 M 个粒子(抗体)x_i 及其速度 v_i,其中 $i=1,\cdots,N$,最后形成初始粒子(抗体)群体 P_0。

(3) 生产免疫记忆粒子(抗体):计算当前粒子(抗体)群体 P 中粒子(抗体)的适应值并判断算法是否满足结束条件,如果满足则结束并输出结果,否则继续运行。

(4) 更新局部和全局最优解,并根据下面公式更新粒子位置和速度:

$$x_{i,j}(t+1)=x_{i,j}(t)+v_{i,j}(t+1),\quad j=1,\cdots,d$$

$$v_{i,j}(t+1)=w\cdot v_{i,j}(t)+c_1r_1[p_{i,j}-x_{i,j}(t)]+c_2r_2[p_{g,j}-x_{i,j}(t)]$$

(5) 由 logistic 映射产生 N 个新的粒子(抗体)。

(6) 基于浓度的粒子(抗体)选择:用群体中相似抗体百分比计算生产 $N+M$ 个新粒子(抗体)的概率,依照概率大小选择 N 个粒子(抗体)形成粒子(抗体)群 P,然后转入第(3)步。

【例 13-2】 使用基于模拟退火的混合粒子群算法,求解下列函数最小值:

$$f(x)=\frac{\cos\sqrt{x_1^2-x_2^2}-3}{[2+(x_1^2+x_2^2)]^2}+0.8$$

其中 $-10\leqslant x_i\leqslant 10$。粒子数为 60,学习因子均为 1.2,退火常数取值 0.8,迭代步数为 800。

解: 根据分析,建立免疫粒子群算法的 MATLAB 代码如下:

```
function [x,y,Result] = PSO_immu(func,N,c1,c2,w,MaxDT,D,eps,DS,replaceP,minD,Psum)
format long;
%%%%%% 给定初始化条件 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c1 = 1.2;                                         % 学习因子 1
c2 = 1.2;                                         % 学习因子 2
w = 0.8;                                          % 惯性权重
MaxDT = 800;                                      % 最大迭代次数
D = 2;                                            % 搜索空间维数(未知数个数)
N = 60;                                           % 初始化群体个体数目
eps = 10 ^( - 10);                                % 设置精度(在已知最小值的时候用)
DS = 8;                                           % 每隔 DS 次循环就检查最优个体是否变优
replaceP = 0.5;                                   % 粒子的概率大于 replaceP 将被免疫替换
minD = 1e - 10;                                   % 粒子间的最小距离
Psum = 0;                                         % 个体最佳的和
range = 100;
count  =  0;
%%%%%% 初始化种群的个体 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = - range + 2 * range * rand;      % 随机初始化位置
        v(i,j) = randn;                           % 随机初始化速度
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% 先计算各个粒子的适应度,并初始化 Pi 和 Pg %%%%%%%%%%%%%%%%%%%
for i = 1:N
    p(i) = feval(func,x(i,:));

    y(i,:) = x(i,:);
end
pg = x(1,:);                                      % pg 为全局最优
for i = 2:N
    if feval(func,x(i,:))< feval(func,pg)
        pg = x(i,:);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% 主循环,按照公式依次迭代,直到满足精度要求 %%%%%%%
for t = 1:MaxDT
    for i = 1:N
        v(i,:) = w * v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if feval(func,x(i,:))< p(i)
            p(i) = feval(func,x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)< feval(func,pg)
            pg = y(i,:);
            subplot(1,2,1);
```

```
                bar(pg,0.25);
                axis([0 3 -40 40 ]) ;
                title (['Iteration ', num2str(t)]); pause (0.1);
                subplot(1,2,2);
              plot(pg(1,1),pg(1,2),'rs','MarkerFaceColor','r', 'MarkerSize',8)
                hold on;
                plot(x(:,1),x(:,2),'k.');
                set(gca,'Color','g')
                hold off;
                grid on;
                axis([ -100 100 -100 100 ]) ;
                title(['Global Min = ',num2str(p(i))]);
                xlabel(['Min_x= ',num2str(pg(1,1)),' Min_y= ',num2str(pg(1,2))]);

            end
        end
        Pbest(t) = feval(func,pg) ;
%       if Foxhole(pg,D)< eps                         %如果结果满足精度要求则跳出循环
%           break;
%       end
%%%%%开始进行免疫%%%%%%%%%%%%%%%%
        if t > DS
            if mod(t,DS) == 0 && (Pbest(t - DS + 1) - Pbest(t))< 1e-020   %如果连续 DS 代数,群
体中的最优没有明显变优,则进行免疫
                %在函数测试的过程中发现,经过一定代数的更新,个体最优不完全相等,但变化非
常非常小
            for i = 1:N                               %先计算出个体最优的和
              Psum = Psum + p(i);
            end

            for i = 1:N                               %免疫程序

                for j = 1:N                           %计算每个个体与个体 i 的距离
                    distance(j) = abs(p(j) - p(i));
                end
                num = 0;
                for j = 1:N                           %计算与第 i 个个体距离小于 minD 的个数
                    if distance(j)< minD
                        num = num + 1;
                    end
                end
                PF(i) = p(N - i + 1)/Psum;            %计算适应度概率
                PD(i) = num/N;                        %计算个体浓度

                a = rand;                             %随机生成计算替换概率的因子
                PR(i) = a * PF(i) + (1 - a) * PD(i);  %计算替换概率
            end

             for i = 1:N
```

```
                    if PR(i)> replaceP
                        x(i,:) = - range + 2 * range * rand(1,D);
                    count = count + 1;
                    end
                end
            end
        end
    end

%%%%%%% 最后给出计算结果 %%%%%%%%%%%%%%%%%%%%
x = pg(1,1);
y = pg(1,2);
Result = feval(func,pg);
%%%%%%%%%% 算法结束 %%%%%%%%%%%%%%%%%%
function probabolity(N,i)
PF = p(N - i)/Psum;                % 适应度概率
disp(PF);
for jj = 1:N
  distance(jj) = abs(P(jj) - P(i));
end
num = 0;
for ii = 1:N
    if distance(ii)< minD
        num = num + 1;
    end
end
PD = num/N;                        % 个体浓度
PR = a * PF + (1 - a) * PD;        % 替换概率
```

建立目标函数 MATLAB 代码如下：

```
function y = imF(x)
   y = (cos(x(1)^2 - x(2)^2) - 3)/((2 + (x(1)^2 + x(2)^2))^2) + 0.8;
end
```

编写目标函数最小值计算代码如下：

```
clear all
clc
x = zeros(1,10);
[x1,x2,f] = PSO_im(@imF,60,2,2,0.8,800,5,0.0000001,10,0.6,0.0000000000000000001,0);
% 得到出计算结果
disp('*************************************************');
disp('目标函数取最小值时的自变量: ');
x1
x2
disp('目标函数的最小值为: ')
f
disp('**************************************************');
```

运行后得到结果为

```
*****************************************************
目标函数取最小值时的自变量:
x1 =
    -3.043794662573965e-09
x2 =
    8.562456641693914e-09
目标函数的最小值为:
f =
   0.300000000000000
*****************************************************
```

13.3 粒子群算法的权重控制

惯性权重控制前一变化量对当前变化量的影响，如果 ω 较大，能够搜索以前未能达到的区域，整个算法的全局搜索能力较强；若 ω 较小，则前一部分的影响较小，主要是在当前解的附近搜索，局部搜索能力较强。

常见的 PSO 算法有自适应权重法、随机权重法和线性递减权重法等。

13.3.1 线性递减法

针对 PSO 算法容易早熟及后期容易在全局最优解附近产生振荡的现象，提出了线性递减权重法，即使惯性权重依照线性从大到小递减，其变化公式为

$$w = w_{\max} - \frac{t \times (w_{\max} - w_{\min})}{t_{\max}}$$

其中，$w_{\max}$表示惯性权重最大值，$w_{\min}$表示惯性权重最小值，t 表示当前迭代步数。

随机权重法的计算步骤如下：

（1）随机设置各个粒子的速度和位置。

（2）评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值 p_{best}中，将所有 p_{best}中最优适应值的个体位置和适应值保存在全局极值 g_{best}中。

（3）更新粒子位移和速度：

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \quad j = 1,\cdots,d$$

$$v_{i,j}(t+1) = w \cdot v_{i,j}(t) + c_1 r_1[p_{i,j} - x_{i,j}(t)] + c_2 r_2[p_{g,j} - x_{i,j}(t)]$$

（4）更新权重

$$w = w_{\max} - \frac{t \times (w_{\max} - w_{\min})}{t_{\max}}$$

（5）将每个粒子的适应值与粒子的最好位置比较，如果相近，则将当前值作为粒子最好的位置。比较当前所有的 p_{best}和 g_{best}，更新 g_{best}。

（6）当算法达到其停止条件时，则停止搜索并输出结果；否则返回到第（3）步继续搜索。

将实现自适应权重的优化函数命名为 PSO_line，在 MATLAB 中编写实现以上步骤的代码如下：

```
function [xm,fv] = PSO_line(fitness,N,c1,c2,wmax,wmin,M,D)
format long;
%fitness 学习函数
%c1 学习因子 1
%c2 学习因子 2
%wmax 惯性权重最大值
%wmin 惯性权重最值小
%M 最大迭代次数
%D 搜索空间维数
%N 初始化群体个体数目
%xm 目标函数取最小值时的自变量
%fv 目标函数最小值
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%初始化种群的个体%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;
        v(i,j) = randn;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%先计算各个粒子的适应度,并初始化 Pi 和 Pg%%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg  =  x(N,:);                %Pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%主循环,按照公式依次迭代%%%%%%%%%%%%%%%%%%%%%%%%
for t = 1:M
    for i = 1:N
        w  =  wmax  -  (t-1) * (wmax-wmin)/(M-1);
        v(i,:) = w * v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:))<p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)<fitness(pg)
            pg = y(i,:);
        end
```

```
        end
        Pbest(t) = fitness(pg);
    end
    % 得到出计算结果
    disp('*************************************************');
    disp('目标函数取最小值时的自变量:');
    xm = pg'
    disp('目标函数的最小值为:')
    fv = fitness(pg)
    disp('*************************************************');
```

【例 13-3】 用随机权重法求解下列函数的最小值：

$$f(x) = 11(x_1^2 - x_2)^2 - (1 - x_2)^2 + 2(1 + x_2)^2 + 0.7$$

其中，粒子数为 60，学习因子均为 1.2，惯性权重取值[0.6，0.9]，迭代步数为 700。

解：建立目标函数 MATLAB 代码如下：

```
function y = LineF(x)
    y = 11 * ((x(1)^2 - x(2))^2) - (1 - x(2))^2 + 2 * (1 + x(2))^2 + 0.7;end
```

编写如下代码：

```
clear all
clc
x = zeros(1,10);
[xm,fv] = PSO_lin(@LineF,60,1.2,1.2,0.9,0.6,700,2);
% 取自变量
xm;
% 取函数最小值
fv;
```

运行后得到结果如下：

```
*************************************************
目标函数取最小值时的自变量:

xm =

   -0.000000001900051
   -0.250000000604457

目标函数的最小值为:

fv =

   0.950000000000000

*************************************************
```

以上结果说明，用线性递减权重的方法得到了精确的最优点。需要注意的是，惯性权重并不是对所有的问题都有效，具体问题需要具体分析最合适的方法。

13.3.2 自适应法

本节主要介绍两种自适应修改权重的方法。

1. 根据全局最优点的距离进行调整

一些学者认为惯性权重的大小还和其距全局最优点的距离有关，并提出了不同粒子惯性权重不仅随迭代次数的增加而递减，还随距全局最优点距离的增加而递增，即权重 w 根据粒子的位置不同而动态变化。目前大多采用的非线性动态惯性权重系数公式，该公式如下：

$$w=\begin{cases} w_{\min}-\dfrac{(w_{\max}-w_{\min})\times(f-f_{\min})}{f_{\text{avg}}-f_{\min}}, & f\leqslant f_{\text{avg}} \\ w_{\max}, & f>f_{\text{avg}} \end{cases}$$

其中，f 表示粒子实时的目标函数值，f_{avg} 和 $f_{\min}$ 分别表示当前所有粒子的平均值和最小目标值。从上面公式可以看出，惯性权重随着粒子目标函数值的改变而改变。当粒子目标值分散时，减小惯性权重；粒子目标值一致时，增加惯性权重。

根据全局最优点的距离调整算法的基本步骤如下：

(1) 随机初始化种群中各个粒子的位置和速度。

(2) 评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值 p_{best} 中，将所有 p_{best} 中最优适应值的个体位置和适应值保存在全局极值 g_{best} 中。

(3) 更新粒子位移和速度：

$$x_{i,j}(t+1)=x_{i,j}(t)+v_{i,j}(t+1),\quad j=1,\cdots,d$$

$$v_{i,j}(t+1)=w\cdot v_{i,j}(t)+c_1r_1[p_{i,j}-x_{i,j}(t)]+c_2r_2[p_{g,j}-x_{i,j}(t)]$$

(4) 更新权重

$$w=\begin{cases} w_{\min}-\dfrac{(w_{\max}-w_{\min})\times(f-f_{\min})}{f_{\text{avg}}-f_{\min}}, & f\leqslant f_{\text{avg}} \\ w_{\max}, & f>f_{\text{avg}} \end{cases}$$

(5) 将每个粒子的适应值与粒子的最好位置比较，如果相近，则将当前值作为粒子最好的位置。比较当前所有的 p_{best} 和 g_{best}，更新 g_{best}。

(6) 当算法达到其停止条件时，则停止搜索并输出结果；否则返回到第(3)步继续搜索。

2. 依据早熟收敛程度和适应值进行调整权重

根据群里的早熟收敛程度和个体适应值，可以确定惯性权重的变化。

设定粒子 p_i 的适应值为 f_i，最优粒子适应度是 f_m，则粒子群的平均适应值是 $f_{\text{avg}}=\dfrac{1}{n}\sum\limits_{i=1}^{n}f_i$，将优于平均适应值的粒子适应值求平均(记为 f'_{avg})，定义 $\Delta=|f_m-f'_{\text{avg}}|$。

依据 f_i、f_m、f_{avg}将群体分为 3 个子群，分别进行不同的自适应操作。其惯性权重的调整如下：

(1) 如果 f_i 优于 f'_{avg}，那么

$$w = w - (w - w_{min}) \cdot \left| \frac{f_i - f'_{avg}}{f_m - f'_{avg}} \right|$$

(2) 如果 f_i 优于 f'_{avg}，且次于 f_m，则惯性权重不变。

(3) 如果 f_i 次于 f'_{avg}，则

$$w = 1.5 - \frac{1}{1 + k_1 \cdot \exp(-k_2 \cdot \Delta)}$$

其中，k_1、k_2 为控制参数，k_1 用来控制 w 的上限，k_2 主要用来控制 $w = 1.5 - \frac{1}{1 + k_1 \cdot \exp(-k_2 \cdot \Delta)}$的调节能力。

当算法停止时，如果粒子的分布分散，则 Δ 比较大，w 变小，此时算法局部搜索能力加强，从而使得群体趋于收敛；若粒子的分布聚集，则 Δ 比较小，w 变大，使粒子具有较强的探查能力，从而有效地跳出局部最优。

将实现自适应权重的优化函数命名为 PSO_adp，在 MATLAB 中编写实现以上步骤的代码如下：

```
function [xm,fv] = PSO_adp(fitness,N,c1,c2,wmax,wmin,M,D)
format long;
%fitness 学习函数
%c1 学习因子 1
%c2 学习因子 2
%wmax 惯性权重最大值
%wmin 惯性权重最值小
%M 最大迭代次数
%D 搜索空间维数
%N 初始化群体个体数目
%xm 目标函数取最小值时的自变量
%fv 目标函数最小值
%初始化种群的个体
for i = 1:N
    for j = 1:D
        x(i,j) = randn;
        v(i,j) = randn;
    end
end
%先计算各个粒子的适应度
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                        %pg 表示全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end
```

```
%进入主要循环
for t = 1:M
    for j = 1:N
        fv(j) = fitness(x(j,:));
    end
    fvag = sum(fv)/N;
    fmin = min(fv);
    for i = 1:N
        if fv(i) <= fvag
            w = wmin + (fv(i) - fmin) * (wmax - wmin)/(fvag - fmin);
        else
            w = wmax;
        end
        v(i,:) = w * v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:))< p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)< fitness(pg)
            pg = y(i,:);
        end
    end
end
%得到出计算结果
disp('*************************************************');
disp('目标函数取最小值时的自变量：');
xm = pg'
disp('目标函数的最小值为：')
fv = fitness(pg)
disp('*************************************************');
```

【例 13-4】 用自适应权重法求解下列函数的最小值。

$$f(x) = \frac{\cos^2 \sqrt{x_1^2 + x_2^2} - \sin \sqrt{x_1^2 + x_2^2} + 2}{[3 + 0.2(x_1^2 + x_2^2)]^2} - 0.9$$

其中，粒子数为 40，学习因子均为 1.5，惯性权重取值[0.7，0.7]，迭代步数为 200。

解：首先建立目标函数代码如下：

```
function y = fitness(x)
  y = ((cos(x(1)^2 - x(2)^2))^2 - sin(x(1)^2 + x(2)^2) + 2)/((3 + 0.2 * (x(1)^2 + x(2)^2))
^2) - 0.9;end
```

编写目标函数最小值计算代码如下：

```
clear all
clc
x = zeros(1,10);
```

```
[xm,fv] = PSO_adp(@fitness,40,1.5,1.5,0.7,0.7,200,2)
%取自变量
xm;
%取函数最小值
fv;
```

运行后得到结果如下：

```
*************************************************
目标函数取最小值时的自变量:
xm =
   1.0e+04 *
   2.930330274608174
   0.507033808260538

目标函数的最小值为:
fv =
  -0.900000000000000
*************************************************
```

13.4 混合粒子群算法

混合策略 PSO 就是将其他进化算法或传统优化算法或其他技术应用到 PSO 中，用于提高局部开发能力、增强收敛速度与精度，或者提高粒子多样性、增强粒子的全局探索能力。

本节主要介绍基于模拟退火和杂交两种混合粒子群算法的 MATLAB 应用。

13.4.1 模拟退火免疫算法

基于杂交的混合粒子群算法步骤如下：

(1) 随机设置各个粒子的速度和位置。

(2) 评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值 p_{best} 中，将所有 p_{best} 中最优适应值的个体位置和适应值保存在全局极值 g_{best} 中。

(3) 确定初始温度。

(4) 根据下式确定当前温度下各粒子 p_i 的适应值

$$\text{TF}(p_i)=\frac{e^{-(f(p_i)-f(p_g))/t}}{\sum_{i=1}^{N}e^{-(f(p_i)-f(p_g))/t}}$$

(5) 从所有 p_i 中确定全局最优的替代值 p_i'，并根据下面两个公式更新各粒子的速位置和速度：

$$x_{i,j}(t+1)=x_{i,j}(t)+v_{i,j}(t+1),\quad j=1,\cdots,d$$
$$v_{i,j}(t+1)=\varphi\{v_{i,j}(t)+c_1r_1[p_{i,j}-x_{i,j}(t)]+c_2r_2[p_{g,j}-x_{i,j}(t)]\}$$

$$\varphi=\frac{2}{2-(c_1+c_2)-\sqrt{(c_1+c_2)^2-4(c_1+c_2)}}$$

(6) 计算粒子目标值，并更新 p_{best} 和 g_{best}，然后进行退温操作。

(7) 当算法达到其停止条件时，则停止搜索并输出结果；否则返回到第(4)步继续搜索。

(8) 初始温度和退温方式对算法有一定的影响，一般采用如下所示的初始温度和退温方式：

$$t_{k+1}=\lambda t_k,\quad t_0=f(p_g)/\ln 5$$

【例 13-5】 使用基于模拟退火的混合粒子群算法，求解下列函数的最小值。

$$f(x)=1/\left[0.9+\sum_{i=1}^{8}\frac{i-3}{(x_i-1)^2+0.8}\right]$$

其中 $-10\leqslant x_i\leqslant 10$。粒子数为 80，学习因子均为 1.8，退火常数取值 0.8，迭代步数为 800。

解：根据分析，建立模拟退火粒子群算法的 MATLAB 代码如下：

```
function [xm,fv] = PSO_moni(fitness,N,c1,c2,lamda,M,D)
format long;
%N 初始化群体个体数目
%c1 学习因子 1
%c2 学习因子 2
%lamda 退火常数惯性权重
%M 最大迭代次数
%D 搜索空间维数
%%%%%%%%%%%初始化种群的个体%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;                    %初始化位置
        v(i,j) = randn;                    %初始化速度
    end
end
%%%%%%%%%%先计算各个粒子的适应度,并初始化 Pi 和 Pg%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                               %pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end
%%%%%主循环,按照公式依次迭代%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T = - fitness(pg)/log(0.2);
for t = 1:M
    groupFit = fitness(pg);
    for i = 1:N
```

```
        Tfit(i) = exp( - (p(i) - groupFit)/T);
    end
    SumTfit = sum(Tfit);
    Tfit = Tfit/SumTfit;
    pBet = rand();
    for i = 1:N
        ComFit(i) = sum(Tfit(1:i));
        if pBet <= ComFit(i)
            pg_plus = x(i,:);
           break;
        end
    end
    C = c1 + c2;
    ksi = 2/abs( 2 - C - sqrt(C^2 - 4 * C));
    for i = 1:N
        v(i,:) = ksi * (v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg_plus - x
(i,:)));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:))< p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)< fitness(pg)
            pg = y(i,:);
        end
    end
    T = T * lamda;
    Pbest(t) = fitness(pg);
end
xm = pg';
fv = fitness(pg);
```

建立目标函数 MATLAB 代码如下：

```
function y = moniF(x)
y = 0;
for i = 1:8
    y = y + (i - 3)/(((x(i) - 1)^2) + 0.8);
end
y = 1/(0.9 + y);
end
```

编写目标函数最小值计算代码如下：

$$f(x) = 1/\left[0.9 + \sum_{i=1}^{8} \frac{i-3}{(x_i-1)^2+0.8}\right]$$

```
clear all
clc
x = zeros(1,10);
[xm,fv] = PSO_moni(@moniF,80,1.8,1.8,0.8,800,8)
%得到出计算结果
disp('*************************************************');
disp('目标函数取最小值时的自变量：');
xm
disp('目标函数的最小值为：')
fv
disp('*************************************************');
```

运行后得到结果如下：

```
*************************************************
目标函数取最小值时的自变量：

xm =
  21.465979942298588
   0.671960645021363
  22.064689929871378
 -29.621543758553962
 -40.816674147029708
  -8.000879764932099
   6.215638420822061
 -13.525875786000029

目标函数的最小值为：
fv =
     -8.098063649737700e+02

*************************************************
```

13.4.2　基于杂交的算法

该算法是借鉴遗传算法中杂交的概念，在每次迭代中，根据杂交率选取指定数量的粒子放入杂交池内，池内的粒子随机两两杂交，产生同样数目的子代粒子(n)，并用子代粒子代替父代粒子(m)。子代位置由父代位置进行交叉得到

$$nx = i \times mx(1) + (1-i) \times mx(2)$$

其中，mx 表示父代粒子的位置，nx 表示子代粒子的位置，i 是 0 到 1 之间的随机数。

子代的速度由下式计算：

$$nv = \frac{mv(1) + mv(2)}{|mv(1) + mv(2)|} |mv|$$

其中，mv 表示父代粒子的速度，nv 表示子代粒子的速度。

基于杂交的混合粒子群算法步骤如下：

(1) 随机设置各个粒子的速度和位置。

(2) 评价每个粒子的适应度,将粒子的位置和适应值存储在粒子的个体极值 p_{best} 中,将所有 p_{best} 中最优适应值的个体位置和适应值保存在全局极值 g_{best} 中。

(3) 更新粒子位移和速度:

$$x_{i,j}(t+1)=x_{i,j}(t)+v_{i,j}(t+1),\quad j=1,\cdots,d$$

$$v_{i,j}(t+1)=w\cdot v_{i,j}(t)+c_1r_1[p_{i,j}-x_{i,j}(t)]+c_2r_2[p_{g,j}-x_{i,j}(t)]$$

(4) 将每个粒子的适应值与粒子的最好位置比较,如果相近,则将当前值作为粒子最好的位置。比较当前所有的 p_{best} 和 g_{best},更新 g_{best}。

(5) 根据杂交概率选取指定数量的粒子,并将其放入杂交池中,池中的粒子随机两两杂交产生同样数目的子代粒子,子代的位置和速度计算公式如下:

$$\begin{cases}nx=i\times mx(1)+(1-i)\times mx(2)\\ mv=\dfrac{mv(1)+mv(2)}{|mv(1)+mv(2)|}|mv|\end{cases}$$

其中,保持 p_{best} 和 g_{best} 不变;

(6) 当算法达到其停止条件时,则停止搜索并输出结果;否则返回到第(3)步继续搜索。

【例 13-6】 使用基于杂交的混合粒子群算法,求解下列函数的最小值。

$$f(x)=1/\left[3+\sum_{i=1}^{8}\frac{i}{1-(x_i+2)^2}\right]+0.8$$

其中 $-8\leqslant x_i\leqslant 8$。粒子数为 80,学习因子均为 1.8,惯性权重取值 0.8,杂交概率为 0.8,杂交池比例为 0.1,迭代步数为 800。

解:根据分析,建立杂交粒子群算法的 MATLAB 代码如下:

```
function [xm,fv] = PSO_br(fitness,N,c1,c2,w,bc,bs,M,D)
format long;
%fitness 学习函数
%N 初始化群体个体数目
%c1 学习因子 1
%c2 学习因子 2
%w 惯性权重
%bc 杂交概率
%bs 杂交池的大小比例
%M 最大迭代次数
%D 搜索空间维数
%xm 目标函数取最小值时的自变量
%fv 目标函数最小值
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%% 初始化种群的个体 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;                    %随机初始化位置
        v(i,j) = randn;                    %随机初始化速度
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%先计算各个粒子的适应度,并初始化Pi和Pg%%%%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                    %pg为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%进入主要循环,按照公式依次迭代%%%%%%%%%%%%%%%%
for t = 1:M
    for i = 1:N
        v(i,:) = w*v(i,:) + c1*rand*(y(i,:) - x(i,:)) + c2*rand*(pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:))<p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)<fitness(pg)
            pg = y(i,:);
        end
        r1 = rand();
        if r1 < bc
            numPool = round(bs*N);
            PoolX = x(1:numPool,:);
            PoolVX = v(1:numPool,:);
            for i = 1:numPool
                seed1 = floor(rand()*(numPool-1)) + 1;
                seed2 = floor(rand()*(numPool-1)) + 1;
                pb = rand();
                childx1(i,:) = pb*PoolX(seed1,:) + (1-pb)*PoolX(seed2,:);
                  childv1(i,:) = (PoolVX(seed1,:) + PoolVX(seed2,:))*norm(PoolVX
(seed1,:))/...
                    norm(PoolVX(seed1,:) + PoolVX(seed2,:));
            end
            x(1:numPool,:) = childx1;
            v(1:numPool,:) = childv1;
        end
    end
end
xm = pg';
fv = fitness(pg);
```

建立目标函数 MATLAB 代码如下：

```
function y = BrF (x)
y = 0;
for i = 1:5
    y = y + i/(1 - (x(i) + 2)^2);
end
y = 1/(3 + y) + 0.8;
end
```

编写目标函数最小值计算代码如下：

```
clear all
clc
x = zeros(1,10);
[xm,fv] = PSO_br (@BrF,50,2,2,0.8,0.8,0.1,100,5);
%得到出计算结果
disp('*************************************************');
disp('目标函数取最小值时的自变量：');
xm
disp('目标函数的最小值为：')
fv
disp('*************************************************');
```

运行后得到结果如下：

```
*************************************************
目标函数取最小值时的自变量:
xm =
   1.026894939358278
   0.222030295553634
   0.063289672557852
   0.254033759001566
   1.417279317699315
  -1.296699233387401
   0.308084174293970
  -4.867674827794006

目标函数的最小值为:
fv =
    -2.414986565354732e+05
*************************************************
```

本章小结

PSO算法起源于对简单社会系统的模拟，具有很好的生物社会背景，易理解、参数少而易实现，对非线性、多峰问题均具有较强的全局搜索能力，在科学研究与工程实践中得到了广泛关注。同时，PSO也是一种很好的优化工具。

粒子群算法思想简单，使用方便。本章首先介绍了粒子群算法的基础，然后介绍了算法的MATLAB实现方法，并介绍了两种典型的权重改进粒子群算法。最后，针对粒子群不同的混合对象，举例说明了粒子群算法的各种混合应用。

第14章 遗传优化算法的实现

遗传算法(genetic algorithm,GA)是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型,是一种通过模拟自然进化过程搜索最优解的方法。

本章重点介绍了基本遗传算法及其 MATLAB 应用,并对 MATLAB 工具箱的应用做了简单介绍。

学习目标:

(1) 了解遗传算法的基本概念;

(2) 掌握 MATLAB 在遗传算法中的编程;

(3) 熟练掌握 MATLAB 遗传算法工具箱的应用。

14.1 遗传算法概述

遗传算法是 1960 年由 Holland 提出来的,其最初的目的是研究自然系统的自适应行为并设计具有自适应功能的软件系统。它的特点是对参数进行编码运算不需要有关体系的任何先验知识,沿多种路线进行平行搜索不会落入局部较优的陷阱。

遗传算法是从代表问题可能潜在的解集的一个种群(population)开始,而一个种群则由经过基因(gene)编码的一定数目的个体(individual)组成。

每个个体实际上是染色体(chromosome)带有特征的实体。染色体作为遗传物质的主要载体,即多个基因的集合,其内部表现(即基因型)是某种基因组合,它决定了个体形状的外部表现,如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此,在一开始需要实现从表现型到基因型的映射,即编码工作。

由于仿照基因编码的工作很复杂,往往需要进行简化,如二进制编码,初代种群产生之后,按照适者生存和优胜劣汰的原理,逐代(generation)演化出越来越好的近似解。

在每一代,根据问题域中个体的适应度(fitness)大小选择(selection)个体,并借助自然遗传学的遗传算子(genetic operators)进行组合交叉(crossover)和变异(mutation),产生出代表新的解集的

种群。

这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境，末代种群中的最优个体经过解码(decoding)，可以作为问题近似最优解。

由于遗传算法的整体搜索策略和优化搜索方法在计算时不依赖于梯度信息或其他辅助知识，而只需要影响搜索方向的目标函数和相应的适应度函数，所以遗传算法提供了一种求解复杂系统问题的通用框架，它不依赖于问题的具体领域，对问题的种类有很强的鲁棒性，所以广泛应用于许多科学。

遗传算法主要应用以下两个领域：

1. 函数优化

函数优化是遗传算法的经典应用领域，也是遗传算法进行性能评价的常用算例，许多人构造出了各种各样复杂形式的测试函数：连续函数和离散函数、凸函数和凹函数、低维函数和高维函数、单峰函数和多峰函数等。对于一些非线性、多模型、多目标的函数优化问题，用其他优化方法较难求解，而遗传算法可以方便地得到较好的结果。

2. 组合优化

随着问题规模的增大，组合优化问题的搜索空间也急剧增大，有时用目前计算的枚举法很难求出最优解。对于这类复杂的问题，人们已经意识到应把主要精力放在寻求满意解上，而遗传算法是寻求这种满意解的最佳工具之一。实践证明，遗传算法对于组合优化中的NP问题非常有效。例如遗传算法已经在求解旅行商问题、背包问题、装箱问题和图形划分问题等方面得到成功的应用。

14.2 基本遗传算法

遗传算法在自然与社会现象模拟和工程计算等方面得到广泛的应用。为了取得更好的效果，在不同的应用领域，人们对遗传算法进行了大量的改进。为了防止混淆，通常把Holland提出的算法称为基本遗传算法。

遗传操作是模拟生物基因遗传的做法。在遗传算法中，通过编码组成初始群体后，遗传操作的任务就是对群体的个体按照它们对环境适应度施加一定的操作，从而实现优胜劣汰的进化过程。从优化搜索的角度而言，遗传操作可使问题的解一代又一代地优化，并逼近最优解。

遗传算法流程图如图14-1所示。

遗传操作包括以下三个基本遗传算子(genetic operator)：选择(selection)、交叉(crossover)、变异(mutation)。

个体遗传算子的操作都是在随机扰动情况下进行的。因此，群体中个体向最优解迁移的规则是随机的。需要强调的是，这种随机化操作和传统的随机搜索方法是有区别的。遗传操作进行高效有向的搜索而不是一般随机搜索方法所进行的无向搜索。

遗传算法不能直接处理问题空间的参数，必须把它们转换成遗传空间中由基因按一定结构组成的染色体或个体。这一转换操作就叫做编码，也可以称作(问题的)表示

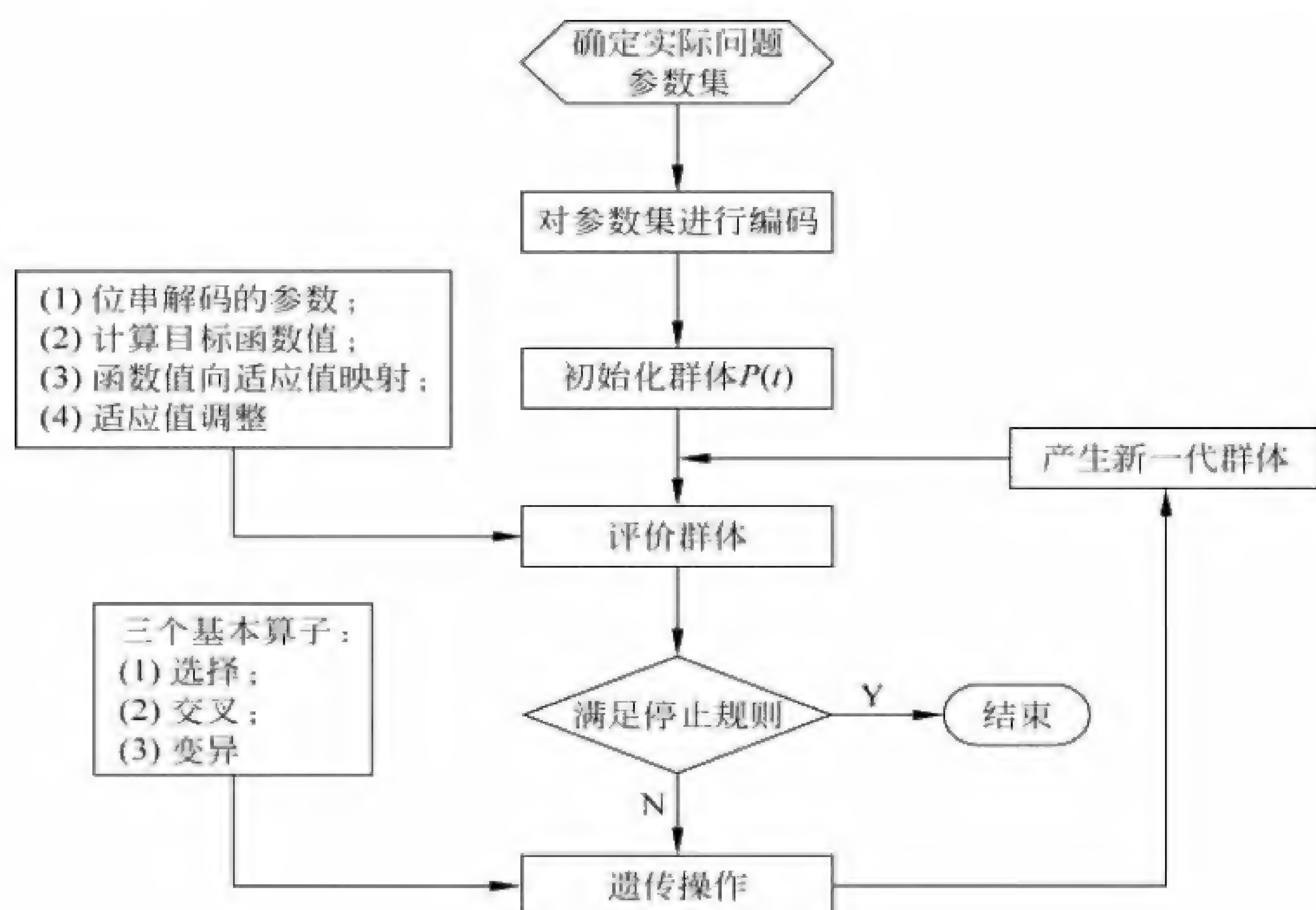


图 14-1　遗传算法流程图

(representation)。

评估编码策略常采用以下三个规范：

(1) 完备性(completeness)：问题空间中的所有点(候选解)都能作为GA空间中的点(染色体)表现。

(2) 非冗余性(nonredundancy)：染色体和候选解一一对应。

(3) 健全性(soundness)：GA空间中的染色体能对应所有问题空间中的候选解。

目前的几种常用的编码技术有二进制编码、浮点数编码、字符编码、变成编码等。

而二进制编码是目前遗传算法中最常用的编码方法，即是由二进制字符集{0,1}产生通常的0,1字符串来表示问题空间的候选解。它具有以下特点：

(1) 简单易行；

(2) 符合最小字符集编码原则；

(3) 便于用模式定理进行分析，因为模式定理就是以此为基础的。

在单纯的遗传算法中，有时候也会出现不收敛的情况，即使在单峰或单调条件下也是如此。这是因为种群的进化能力已经基本丧失，种群早熟。为了避免种群早熟，参数的设计一般遵从以下原则：

(1) 种群的规模：当群体规模太小时，很明显会出现近亲交配，产生病态基因，而且造成有效等位基因先天缺乏，即使采用较大概率的变异算子，生成具有竞争力的高阶模式的可能性仍很小，况且大概率变异算子对已有模式的破坏作用极大。

同时遗传算子存在随机误差(模式采样误差)，妨碍小群体中有效模式的正确传播，使得种群进化不能按照模式定理产生所预测的期望数量；种群规模太大，结果难以收敛且浪费资源，稳健性下降。种群规模的一个建议值为0～100。

(2) 变异概率：当变异概率太小时，种群的多样性下降太快，容易导致有效基因的迅

速丢失且不容易修补；当变异概率太大时，尽管种群的多样性可以得到保证，但是高阶模式被破坏的概率也随之增大。变异概率一般取0.0001～0.2。

(3) 交配概率：交配是生成新种群最重要的手段。与变异概率类似，交配概率太大容易破坏已有的有利模式，随机性增大，容易错失最优个体；交配概率太小不能有效更新种群。交配概率一般取0.4～0.99。

(4) 进化代数：进化代数太小，算法不容易收敛，种群还没有成熟；代数太大，算法已经熟练或者种群过于早熟不可能再收敛，继续进化没有意义，只会增加时间开支和资源浪费。进化代数一般取100～500。

(5) 种群初始化：初始种群的生成是随机的；在初始种群赋予之前，尽量进行一个大概的区间估计，以免初始种群分布在远离全局最优解的编码空间，导致遗传算法的搜索范围受到限制，同时也为算法减轻负担。

遗传算法在搜索进化过程中一般不需要其他外部信息，仅用评估函数来评估个体或解的优劣，并作为以后遗传操作的依据。由于遗传算法中，适应度函数要比较排序并在此基础上计算选择概率，所以适应度函数的值要取正值。由此可见，将目标函数映射成求最大值形式且函数值非负的适应度函数是必要的。

适应度函数的设计主要满足以下条件：

(1) 单值、连续、非负、最大化；

(2) 合理、一致性；

(3) 计算量小；

(4) 通用性强。

在具体应用中，适应度函数的设计要结合求解问题本身的要求而定。适应度函数设计直接影响到遗传算法的性能。

随机初始化种群 $P(t)=\{x_1,x_2,\cdots,x_n\}$，计算 $P(t)$ 中个体的适应值。其MATLAB程序的基本格式如下：

```
Begin
t = 0
初始化 P(t)
计算 P(t)的适应值
while (不满足停止准则)
      do
      begin
      t = t + 1
      从 P(t + 1)中选择 P(t)
      重组 P(t)
计算 P(t)的适应值
end
```

【例 14-1】 求下列函数的最大值。

$f(x)=11\sin(7x)+7\cos(3x)$，其中 $x\in[0,11]$

解：根据基本遗传算法原理，编写MATLAB代码如下：

```
%初始化
function pop = initpop(popsize,chromlength)
pop = round(rand(popsize,chromlength));
%rand 随机产生每个单元为{0,1}行数为 popsize,列数为 chromlength 的矩阵
%roud 对矩阵的每个单元进行圆整.这样产生的初始种群
End

function pop2 = decodebinary(pop)
[px,py] = size(pop);
%求 pop 行和列数
for i = 1:py
pop1(:,i) = 2.^(py-i).*pop(:,i);
end
pop2 = sum(pop1,2);
%求 pop1 的每行之和
end

%将二进制编码转换成十进制
function pop2 = decodechrom(pop,spoint,length)
pop1 = pop(:,spoint:spoint + length - 1);
pop2 = decodebinary(pop1);
end

function [objvalue] = calobjvalue(pop)
temp1 = decodechrom(pop,1,10);           %将 pop 每行转化成十进制数
x = temp1 * 10/1023;                     %将二值域中的数转化为变量域的数
objvalue = 10 * sin(5 * x) + 7 * cos(4 * x);%计算目标函数值
end

%计算个体的适应值
function fitvalue = calfitvalue(objvalue)
global Cmin;
Cmin = 0;
[px,py] = size(objvalue);
for i = 1:px
    if objvalue(i) + Cmin > 0
        temp = Cmin + objvalue(i);
    else
        temp = 0.0;
    end
    fitvalue(i) = temp;
end
fitvalue = fitvalue'

%选择复制
function [newpop] = selection(pop,fitvalue)
totalfit = sum(fitvalue);                %求适应值之和
fitvalue = fitvalue/totalfit;            %单个个体被选择的概率
fitvalue = cumsum(fitvalue);       %如 fitvalue = [1 2 3 4],则 cumsum(fitvalue) = [1 3 6 10]
```

```
[px,py] = size(pop);
ms = sort(rand(px,1));          % 从小到大排列
fitin = 1;
newin = 1;
while newin <= px
    if(ms(newin))< fitvalue(fitin)
        newpop(newin) = pop(fitin);
        newin = newin + 1;
    else
        fitin = fitin + 1;
    end
end

% 交叉
function [newpop] = crossover(pop,pc)
[px,py] = size(pop);
newpop = ones(size(pop));
for i = 1:2:px - 1
    if(rand < pc)
        cpoint = round(rand * py);
        newpop(i,:) = [pop(i,1:cpoint),pop(i + 1,cpoint + 1:py)];
        newpop(i + 1,:) = [pop(i + 1,1:cpoint),pop(i,cpoint + 1:py)];
    else
        newpop(i,:) = pop(i);
        newpop(i + 1,:) = pop(i + 1);
    end
end

% 变异
function [newpop] = mutation(pop,pm)
[px,py] = size(pop);
newpop = ones(size(pop));
for i = 1:px
    if(rand < pm)
        mpoint = round(rand * py);
        if mpoint <= 0
            mpoint = 1;
        end
        newpop(i) = pop(i);
        if any(newpop(i,mpoint)) == 0
            newpop(i,mpoint) = 1;
        else
            newpop(i,mpoint) = 0;
        end
    else
        newpop(i) = pop(i);
    end
end
```

```
%求出群体中适应值最大的值
function [bestindividual,bestfit] = best(pop,fitvalue)
[px,py] = size(pop);
bestindividual = pop(1,:);
bestfit = fitvalue(1);
for i = 2:px
    if fitvalue(i)> bestfit
        bestindividual = pop(i,:);
        bestfit = fitvalue(i);
    end
end

%主程序
clear all
clc
popsize = 20;                                    %群体大小
chromlength = 10;                                %字符串长度(个体长度)
pc = 0.8;                                        %交叉概率
pm = 0.006;                                      %变异概率
pop = initpop(popsize,chromlength);              %随机产生初始群体
for i = 1:30                                     %30 为迭代次数
[objvalue] = calobjvalue(pop);                   %计算目标函数
fitvalue = calfitvalue(objvalue);                %计算群体中每个个体的适应度
[newpop] = selection(pop,fitvalue);              %复制
[newpop] = crossover(pop,pc);                    %交叉
[newpop] = mutation(pop,pc);                     %变异
[bestindividual,bestfit] = best(pop,fitvalue);   %求出群体中适应值最大的个体及其适应值
y(i) = max(bestfit);
n(i) = i;
pop5 = bestindividual;
x(i) = decodechrom(pop5,1,chromlength) * 10/1023;
pop = newpop;
end
fplot('11 * sin(7 * x) + 7 * cos(3 * x)',[0 11])
hold on
plot(x,y,'r * ')
hold off
f = max(y)
```

运行 MATLAB 程序后，得到结果如下：

```
f =

   17.2830
```

即函数最大值为 17.2830，仿真结果如图 14-2 所示。

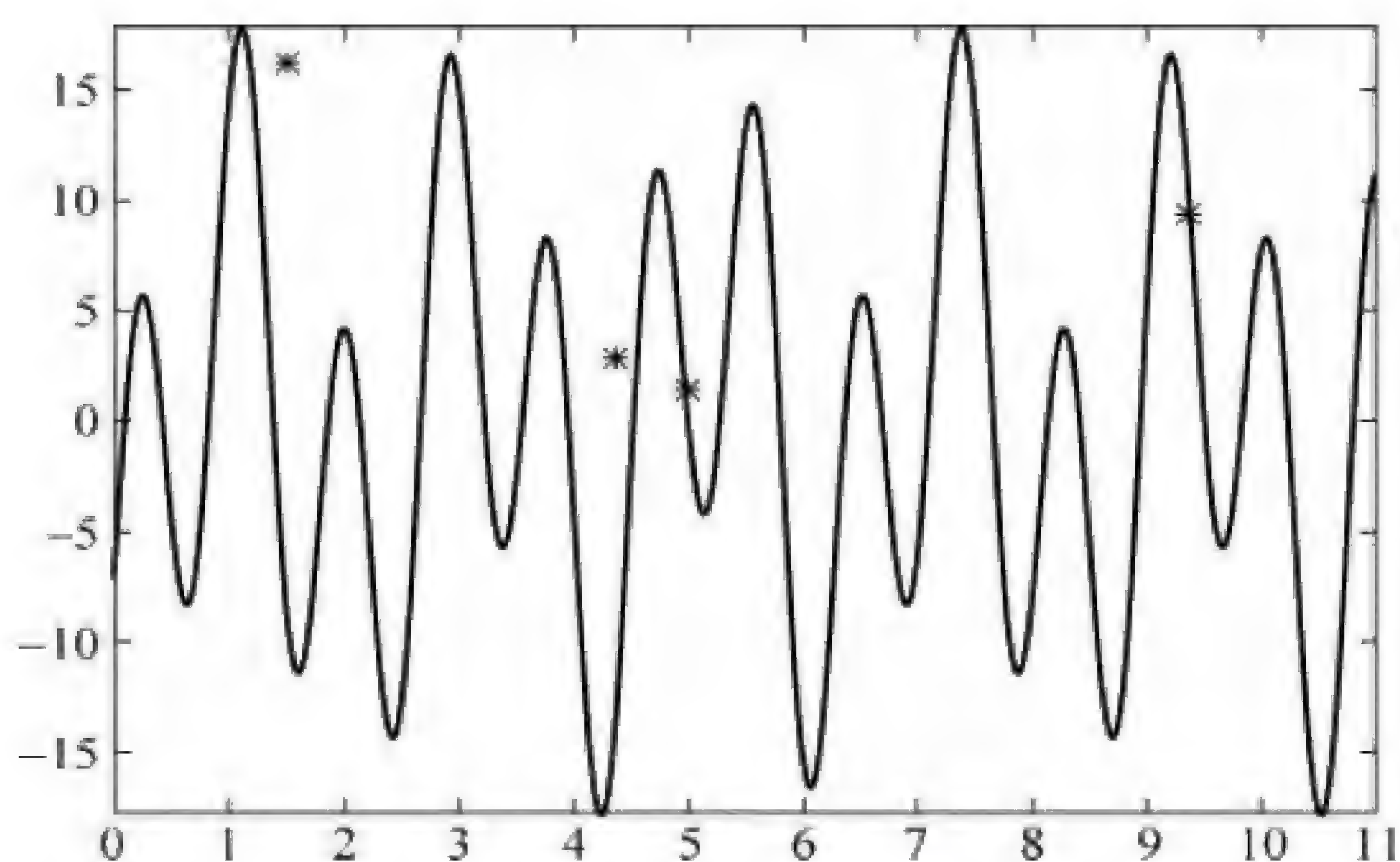


图 14-2　函数值变化图

14.3　MATLAB 遗传算法工具箱及其应用

在 MATLAB 软件中，已经将遗传算法命令进行了封装，做成专门的遗传算法工具箱(GA toolbox)，方便用户调用。

目前 MATLAB 自带的遗传算法与直接搜索工具箱(genetic algorithm and direct search toolbox)可以来优化目标函数。

遗传算法与直接搜索工具箱有 ga、gaoptimset 和 gaoptimget 三个核心函数。

1. ga 函数

ga 函数是对目标函数进行遗传计算，其格式如下：

```
[x,fval,exitflag,output,population,scores] = ga(fitnessfcn,nvars,...,options)
```

其中，fitnessfun 为适应度句柄函数；nvars 为目标函数自变量的个数；options 为算法的属性设置，该属性是通过函数 gaoptimset 赋予的；x 为经过遗传进化以后自变量最佳染色体返回值；fval 为最佳染色体的适应度；exitflag 为算法停止的原因；output 为输出的算法结构；population 为最终得到种群适应度的列向量；scores 为最终得到的种群。

【例 14-2】 使用 ga 函数求解以下不等式的解 x_1、x_2。

$$\begin{bmatrix} -1 & 2 \\ 1 & 3 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leqslant \begin{bmatrix} 3 \\ 5 \\ 2 \end{bmatrix}, x_1 \geqslant 0, x_1 \geqslant 0$$

解：根据函数 ga 的用法，在 MATLAB 中编写代码如下：

```
clear all
clc
A = [-1 2;
        1 3;
        5 1];
```

```
b = [3; 5; 2];lb = zeros(2,1);
[x,fval,exitflag] = ga(@lincontest6,2,A,b,[],[],lb)
```

运行得到结果如下：

```
Optimization terminated: average change in the fitness value less than options.
FunctionTolerance.

x =
      0.0910   1.5460
fval =
   -7.2044
exitflag =
     1
```

从以上结果可知，$x_1=0.091$，$x_2=1.546$。

【例 14-3】 使用遗传算法求解以下函数的最大值。

$$f(x,y)=(\cos(x^2+y^2)-0.8)/(3+0.8\,(x^2+y^2)^2)+8$$

解：编写适应度函数 MATLAB 代码如下：

```
function y = sy (x)
    y = (cos(x(1)^2 + x(2)^3) - 0.8)/(3 + 0.8 * (x(1)^2 + x(2)^2)^2) + 8;
end
```

在 MATLAB 命令行窗口输入代码如下：

```
clear all
clc
[x,fval,exitflag,output,population,scores] = ga(@sy,2)
```

得到结果如下：

```
Optimization terminated: average change in the fitness value less than options.
FunctionTolerance.

x =
    1.5655   -0.0144
fval =
    7.7988
exitflag =
     1
output =

  包含以下字段的 struct:
      problemtype: 'unconstrained'
         rngstate: [1×1 struct]
      generations: 77
        funccount: 3900
```

```
          message: 'Optimization terminated: average change in the fitness value less than
options.FunctionTolerance.'
    maxconstraint: []

population =


    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -1.1754
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
   -0.7086   12.5581
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
   -2.6471   -0.0144
   -5.1778  -10.2914
    1.5655   -0.0144
  -24.3041   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655    7.7481
    1.5655   -0.0144
    1.5655    2.9799
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
   27.0278   -0.0144
    1.5655   -0.0144
    1.5655   -0.0144
    0.7493   -0.0144
    1.5655   -6.7633
    1.5655   -0.0144
```

```
     1.5655    -0.0144
    -1.9437     7.4166
    18.9690    11.5355
    -7.3073   -18.4697
    -6.9048   -11.9800
    10.1626     3.0646
    12.0223   -14.9907
   -17.3207    14.7323
   -11.7268     5.2566
     1.3252     8.0299


scores =

     7.7988
     7.7988
     7.7988
     7.7988
     7.7988
     7.7988
     7.9917
     7.7988
     7.7988
     7.7988
     7.9999
     7.7988
     7.7988
     7.7988
     7.7988
     7.7988
     7.7988
     7.9988
     8.0000
     7.7988
     8.0000
     7.7988
     7.7988
     7.7988
     7.7988
     7.9995
     7.7988
     7.9848
     7.7988
     7.7988
     7.7988
     7.7988
     7.7988
     7.7988
     8.0000
```

```
    7.7988
    7.7988
    8.0143
    7.9999
    7.7988
    7.7988
    7.9994
    8.0000
    8.0000
    8.0000
    8.0000
    8.0000
    8.0000
    8.0000
    7.9997
```

即当 $x=1.5655$，$y=-0.0144$ 时函数取得最大值为 7.7988。

2. gaoptimset 函数

gaoptimset 函数是设置遗传算法的参数和句柄函数，其使用格式如下：

```
options = gaoptimset('param1',value1,'param2',value2,...)
```

由于遗传算法本质上是一种启发式的随机运算，算法程序经常重复运行多次才能得到理想结果。鉴于此，可以将前一次运行得到的最后种群作为下一次运行的初始种群，如此操作会得到更好的结果。

```
[x,fval,reason,output,final_pop] = ga(@fitnessfcn,nvars);
```

最后一个输出变量 final_pop 返回的就是本次运行得到的最后种群，再将 final_pop 作为 ga 函数的初始种群，语法格式如下：

```
options = gaoptimset('InitialPopulation',finnal_pop);
[x,fval,reason,output,finnal_pop2] = ga(@fitnessfcn,nvars,options);
```

遗传算法和直接搜索工具箱中的 ga 函数是求解目标函数的最小值，所以求目标函数最小值的问题，可直接令目标函数为适应度函数。

【例 14-4】 已知适应度函数 zp，求解函数的最小值。

```
function f = zp(m)
a = [0 49 98 147 196 294 391 489 587 685];
y1 = [6.39 9.48 12.46 14.33 17.10 21.94 22.64 21.43 22.07 24.53];
n = size(a,2);      %1 表示行数,2 表是列数
f = 0;
for i = 1:n
```

```
    y = m(1) * (a(i) + m(2))/(m(3) + a(i) + m(2));
   f = f + ((y1(i) - y).^2);
end
```

解：根据适应度函数中的目标函数，编写最优解求解代码如下：

```
clear all
clc
option = gaoptimset('PopulationSize',100,'Generations',5000,'PlotFcns',@gaplotbestf);
% 种群数 100,迭代数 5000
ga(@zp,3,option)
```

运行程序得到结果如下：

```
  Optimization terminated: average change in the fitness value less than options.
  FunctionTolerance.

  ans =

     30.2532   42.9619   183.9353
```

3. gaoptimget 函数

该函数用于得到遗传算法参数结构中的参数具体值。其调用格式如下：

```
val = gaoptimget(options, 'name')
```

其中，options 为结构体变量，name 为需要得到的参数名称，返回值 val。

14.4 自适应遗传算法

自适应遗传算法是根据当前群体适应度情况，自动修改交叉概率和变异概率，在保护最优个体的同时，加快淘汰较差个体，尽快求得最优解。

下面通过举例，说明自适应遗传算法的 MATLAB 应用。

【例 14-5】 利用自适应遗传算法，求解以下函数最优值。

$$f(x) = x\sin(8x) + 3, \quad x \in [-1, +2]$$

解：根据自适应遗传算法，编写 MATLAB 代码如下：

```
clear all
clc
global chrom lchrom oldpop newpop varible fitness popsize sumfitness    % 定义全局变量
global pcross pmutation temp bestfit maxfit gen bestgen
global maxgen po pp mp np
```

```
lchrom = 18;                              % 染色体长度
popsize = 80;                             % 种群大小
pcross = 0.8;                             % 交叉概率
pmutation = 0.02;                         % 变异概率
maxgen = 200;                             % 最大代数
po = 0.1;                                 % 淘汰概率
pp = 0.1;                                 % 保护概率
mp = floor(pp * popsize);                 % 保护的个数
np = floor(po * popsize);                 % 淘汰的个数
initpop;                                  % 初始化种群
for gen = 1:maxgen
      objfun;                             % 计算适应度值
      pp_po;                              % 执行保优操作
      select;                             % 选择操作
      selfmutation;                       % 自变异操作
      crossover;                          % 交叉操作
end
best
bestfit                                   % 最佳个体适应度值输出
bestgen                                   % 最佳个体所在代数输出
figure
gen = 1:maxgen;
plot(gen,maxfit(1,gen));                  % 进化曲线
hold on;
plot(bestgen,bestfit);
xlabel('Generation');
ylabel('Fitness');

% 产生初始种群
function initpop()
global lchrom oldpop popsize chrom
for i = 1:popsize
    chrom = rand(1,lchrom);               % lchrom = 12 染色体长度
    for j = 1:lchrom
        if chrom(1,j)<0.5
            chrom(1,j) = 0;
       else
           chrom(1,j) = 1;
        end
    end
    oldpop(i,1:lchrom) = chrom;
end

% 计算适应度值
function objfun()
global lchrom oldpop fitness popsize chrom maxfit gen varible avgfiness savgfitness
a = 0;
b = 10;
for i = 1:popsize
```

```
    chrom = oldpop(i,:);
    c = decimal(chrom);
    varible(1,i) = a + c * (b - a)/(2.^lchrom - 1);      % 对应变量值
    fitness(1,i) = varible(1,i) * sin(8 * varible(1,i)) + 3;
end
avgfitness = sum(fitness)/popsize;
lsort;                                              % 个体排序
maxfit(1,gen) = max(fitness);                       % 求本代中的最大适应度值 maxfit

%二进制转十进制
function c = decimal(chrom)
global lchrom popsize
c = 0;
for j = 1:lchrom
    c = c + chrom(1,j) * 2.^(lchrom - j);
end

%个体从小到大排序
function lsort()
global popsize fitness oldpop
for i = 1:popsize
    j = i + 1;
    while j <= popsize
        if fitness(1,i)> fitness(1,j)
            tf = fitness(1,i);                      % 适应度值
            tc = oldpop(i,:);                       % 基因代码
            fitness(1,i) = fitness(1,j);            % 适应度值互换
            oldpop(i,:) = oldpop(j,:);              % 基因代码互换
            fitnescs(1,j) = tf;
            oldpop(j,:) = tc;
        end
        j = j + 1;
    end
end

%保优操作
function pp_po()
global popsize oldpop np
i = np + 1;          % np = floor(po * popsize); % 淘汰的个数 np
j = 1;
while i <= popsize   % 将(np + 1)~popsize 的个体放在 toldpop 中,共(popsize - np)个
    toldpop(j,:) = oldpop(i,:);
    j = j + 1;
    i = i + 1;
end
for i = 1:(popsize - np)               % 从小到大顺序排列,将前面 np 个淘汰
    oldpop(i,:) = toldpop(i,:);        % 适应度是否也要互换
End

%转轮法选择操作
```

```
function select()
global fitness popsize sumfitness oldpop temp mp np
sumfitness = 0;                    % 个体适应度之和
for i = 1:(popsize - np - mp)      % 仅计算(popsize - np - mp)个个体的选择概率
    sumfitness = sumfitness + fitness(1,i);
end
for i = 1:(popsize - mp - np)      % 仅计算(popsize - np - mp)个个体的选择概率
    p(1,i) = fitness(1,i)/sumfitness;   % 个体染色体的选择概率
end
q = cumsum(p);                     % 个体染色体的累积概率(内部函数),共(popsize - np - mp)个
b = sort(rand(1,(popsize - mp)));   % 产生(popsize - mp)个随机数,并按升序排列.mp 为保护个
                                      体数
j = 1;
k = 1;
while j <= (popsize - mp)   % 从(popsize - mp - np)中选出(popsize - mp)个个体,并放入 temp
                              (j,:)中
    if b(1,j)< q(1,k)
       temp(j,:) = oldpop(k,:);
       j = j + 1;
    else
       k = k + 1;
    end
end
j = popsize - np - mp + 1;   % 从统一挪过来的(popsize - np - mp)以后个体——优秀个体中选择
for i = (popsize - mp + 1):popsize   % 将 mp 个保留个体放入交配池 temp(i,:),以保证群体
                                       数 popsize
    temp(i,:) = oldpop(j,:);
    j = j + 1;
end

% 自适应变异操作
function selfmutation()
global i popsize lchrom pmutation temp newpop oldpop mp fitness avgfitness maxfitness
m = lchrom * (popsize - mp);                                  % 总的基因数
pm1 = pmutation;
pm2 = 0.005;
a = 0;
b = 10;
for i = 1:popsize
    chrom = oldpop(i,:);
    c = decimal(chrom);
    varible(1,i) = a + c * (b - a)/(2.^lchrom - 1);              % 对应变量值
    fitness(1,i) = varible(1,i) * sin(10 * varible(1,i)) + 2;    % 目标函数
    if(fitness(1,i)>= avgfitness)
     pmutatio = pm1 - (pm1 - pm2) * (fitness(1,i) - avgfitness)/(maxfitness - avgfitness);
      else
          pmutation = pm1;
    end
end
```

```
n = round(pmutation * m);                    % 变异发生的次数
for i = 1:n                                  % 执行变异操作循环
    k = round(rand * (m - 1)) + 1;           % 确定变异位置(四舍五入取整)
    j = ceil(k/lchrom);                      % 确定个体编号(取整)
    l = rem(k,lchrom);                       % 确定个体中变位基因的位置(求余)
    if l == 0
       temp(j,lchrom) = ~temp(j,lchrom);     % 取非操作
    else
       temp(j,l) = ~temp(j,l);               % 取非操作
    end
end
for i = 1:popsize
    newpop(i,:) = temp(i,:);                 % 产生新的个体
    oldpop(i,:) = newpop(i,:);
end
%交叉操作
function crossover()
global temp popsize pcross lchrom mp
n = floor(pcross * (popsize - mp));          % 交叉发生的次数(向下取整)
if rem(n,2)~ = 0                             % 求余
    n = n + 1;                               % 保证为偶数个个体,便于交叉操作
end
j = 1;
m = 0;
%对(popsize - mp)个个体将进行随机配对,满足条件者将进行交叉操作(按顺序选择要交叉的对
象)
for i = 1:(popsize - mp)
    p = rand;                                % 产生随机数
    if p < pcross                            % 满足交叉条件
       parent(j,:) = temp(i,:);              % 选出 1 个父本
       k(1,j) = i;
       j = j + 1;                            % 记录父本个数
       m = m + 1;                            % 记录杂交次数
       if (j == 3)&(m <= n)              % 满足两个父本(j == 3),未超过交叉次数(m <= n)
           pos = round(rand * (lchrom - 1)) + 1; % 确定随机位数(四舍五入取整)
           for i = 1:pos
               child1(1,i) = parent(1,i);
               child2(1,i) = parent(2,i);
           end
           for i = (pos + 1):lchrom
               child1(1,i) = parent(2,i);
               child2(1,i) = parent(1,i);
           end
           i = k(1,1);
           j = k(1,2);
           temp(i,:) = child1(1,:);
           temp(j,:) = child2(1,:);
           j = 1;
```

```
            end
        end
    end

    %最佳个体
    function best()
    global maxfit bestfit gen maxgen bestgen
    bestfit = maxfit(1,1);
    gen = 2;
    while gen <= maxgen
            if bestfit < maxfit(1,gen)
                bestfit = maxfit(1,gen);
                bestgen = gen;
            end
            gen = gen + 1;
    end
```

运行程序得到结果如下：

```
bestfit =
    9.1447

bestgen =
    93
```

即经过93次迭代，函数最优适应值解为9.1447。

14.5 遗传算法的典型应用

遗传算法是基于达尔文的适者生存、优胜劣汰的原理。其具有以下几个突特点：

(1) 对多解的优化问题没有太多的数学要求；

(2) 能有效地进行全局搜索；

(3) 对各种特殊问题有很强的灵活性，应用广泛。

本节将重点阐述遗传算法的两种典型应用。

14.5.1 求解函数极值

本小节通过一个求解简单函数的最小值点的问题来初步展示遗传算法的具体实现方法。

【例14-6】 利用遗传算法求函数 $f(x)=20+x\sin(7\pi x)$，$x\in[-1,1]$的最大值点。

解：在MATLAB中编写绘制函数曲线的代码如下：

```
clear all;
clc;
global BitLength              % 全局变量,计算如果满足求解精度至少需要编码的长度
global boundsbegin            % 全局变量,自变量的起始点
global boundsend              % 全局变量,自变量的终止点
bounds = [ - 1 1];            % 一维自变量的取值范围
precision = 0.0001;           % 运算精度
boundsbegin = bounds(:,1);
boundsend = bounds(:,2);      % 计算如果满足求解精度至少需要多长的染色体
BitLength = ceil(log2((boundsend - boundsbegin)'./ precision));
popsize = 50;                 % 初始种群大小
Generationnmax = 15;          % 最大代数
pcrossover = 0.80;            % 交配概率
pmutation = 0.08;             % 变异概率
population = round(rand(popsize,BitLength));    % 初始种群,行代表一个个体,列代表不同
                                                  个体
%计算适应度
[Fitvalue,cumsump] = fitnessfun(population);   % 输入群体 population,返回适应度 Fitvalue
                                                 和累积概率 cumsump
Generation = 1;
while Generation < Generationnmax + 1
    for j = 1:2:popsize       % 1 对 1 对的群体进行如下操作(交叉,变异)
        %选择
        seln = selection(population,cumsump);
        %交叉
        scro = crossover(population,seln,pcrossover);
        scnew(j,:) = scro(1,:);
        scnew(j + 1,:) = scro(2,:);
        %变异
        smnew(j,:) = mutation(scnew(j,:),pmutation);
        smnew(j + 1,:) = mutation(scnew(j + 1,:),pmutation);
    end
    %产生了新的种群
    population = smnew;

    %计算新种群的适应度
    [Fitvalue,cumsump] = fitnessfun(population);   %记录当前代最好的适应度和平均适应度
    [fmax,nmax] = max(Fitvalue);    %最好的适应度为 fmax(即函数值最大),其对应的个体
                                     为 nmax
    fmean = mean(Fitvalue);     %平均适应度为 fmean
    ymax(Generation) = fmax;    %每代中最好的适应度
    ymean(Generation) = fmean;  %每代中的平均适应度
    %记录当前代的最佳染色体个体
    x = transform2to10(population(nmax,:));   %population(nmax,:)为最佳染色体个体
    xx = boundsbegin + x * (boundsend - boundsbegin)/(power(2,BitLength) - 1);
    xmax(Generation) = xx;
    Generation = Generation + 1
end
Generation = Generation - 1;
```

```
%Generation 加 1、减 1 的操作是为了能记录各代中的最佳函数值 xmax(Generation)
targetfunvalue = targetfun(xmax)
[Besttargetfunvalue,nmax] = max(targetfunvalue)
Bestpopulation = xmax(nmax)
%绘制经过遗传运算后的适应度曲线
figure(2);
hand1 = plot(1:Generation,ymax);
set(hand1,'linestyle','-','linewidth',1,'marker','*','markersize',8)
hold on;
hand2 = plot(1:Generation,ymean);
set(hand2,'color','k','linestyle','-','linewidth',1, 'marker','h','markersize',8)
xlabel('进化代数');
ylabel('最大和平均适应度');
xlim([1 Generationnmax]);
legend('最大适应度','平均适应度');
box off;
hold off;

%计算适应度函数
[Fitvalue,cumsump] = fitnessfun(population);
global BitLength
global boundsbegin
global boundsend
popsize = size(population,1);                          %计算个体个数
for i = 1:popsize
    x = transform2to10(population(i,:));               %将二进制转换为十进制
    %转化为[-2,2]区间的实数
    xx = boundsbegin + x * (boundsend - boundsbegin)/(power(2,BitLength) - 1);
    Fitvalue(i) = targetfun(xx);                       %计算函数值,即适应度
end
%给适应度函数加上一个大小合理的数以便保证种群适应值为正数
Fitvalue = Fitvalue'   %该处还有一个作用就是决定适应度是有利于选取几个有利个体(加强竞
                        争),海深减弱竞争
%计算选择概率
fsum = sum(Fitvalue)
Pperpopulation = Fitvalue/fsum                         %适应度归一化,及被复制的概率
%计算累积概率
cumsump(1) = Pperpopulation(1)
for i = 2:popsize
    cumsump(i) = cumsump(i - 1) + Pperpopulation(i)    %求累计概率
end
cumsump = cumsump'                                     %累计概率

%计算目标函数
function y = targetfun(x);                             %目标函数
y = 20 + x. * sin(7 * pi * x);
end

%新种群交叉操作
```

```
%输入 population 为种群,seln 为选择的两个个体,pc 为交配的概率
function scro = crossover(population,seln,pc);
    BitLength = size(population,2);                  %二进制数的个数
    pcc = IfCroIfMut(pc);      %根据交叉概率决定是否进行交叉操作,1 则是,0 则否
    %进行交叉操作
    if pcc == 1                                      %进行交叉操作
        chb = round(rand * (BitLength - 2)) + 1;     %随机产生一个交叉位
        scro(1,:) = [population(seln(1),1:chb) population(seln(2),chb + 1:BitLength)];
        %序号为 seln(1)的个体在交叉位 chb 前面的信息与序号为 seln(2)的个体在交叉位 chb
+1 后面的信息重新组合
        scro(2,:) = [population(seln(2),1:chb) population(seln(1),chb + 1:BitLength)];
        %序号为 seln(2)的个体在交叉位 chb 前面的信息与序号为 seln(1)的个体在交叉位 chb
+1 后面的信息重新组合
    else
        %不进行交叉操作
        scro(1,:) = population(seln(1),:);
        scro(2,:) = population(seln(2),:);
    end
end

%判断遗传运算是否需要进行交叉或变异
%mutORcro 为交叉、变异发生的概率
%根据 mutORcro 决定是否进行相应的操作,产生 1 的概率是 mutORcro,产生 0 的概率为 1
- mutORcro
function pcc = IfCroIfMut(mutORcro);
test(1:100) = 0;                                     %1×100 的行向量
l = round(100 * mutORcro);              %产生一个数为 100 * mutORcro,round 为取靠近的整数
test(1:l) = 1;
n = round(rand * 99) + 1;
pcc = test(n);
end

%新种群变异操作
%snew 为一个个体
function snnew = mutation(snew,pmutation);
    BitLength = size(snew,2);
    snnew = snew;
    pmm = IfCroIfMut(pmutation);          %根据变异概率决定是否进行变异操作,1 则是,0 则否
    if pmm == 1
        chb = round(rand * (BitLength - 1)) + 1;    %在[1,BitLength]范围内随机产生一个变
                                                      异位
        snnew(chb) = abs(snew(chb) - 1);           %0 变成 1,1 变成 0
    end
end

%选择两个个体,返回个体的序号,有可能两个序号相同
function seln = selection(population,cumsump);   %从种群中选择两个个体
    for i = 1:2
        r = rand;                                    %产生一个随机数
```

```
            prand = cumsump - r;                        % 求出 cumsump 中第一个比 r 大的元素
            j = 1;
            while prand(j)< 0
                j = j + 1;
            end
            seln(i) = j;                                % 选中个体的序号
        end
    end

    % 将二进制数转换为十进制数
    function x = transform2to10(Population);
        BitLength = size(Population,2);                 % Population 的列,即二进制的长度
        x = Population(BitLength);
        for i = 1:BitLength - 1
            x = x + Population(BitLength - i) * power(2,i); % 从末位加到首位
        end
    end
```

运行程序后得到结果如下:

```
Generation =
     2
Generation =
     3
Generation =
     4
Generation =
     5
Generation =
     6
Generation =
     7
Generation =
     8
Generation =
     9
Generation =
    10
Generation =
    11
Generation =
    12
Generation =
    13
Generation =
    14
Generation =
    15
Generation =
```

```
    16
targetfunvalue =
  1 至 13 列
   21.4296  21.4544  21.4544  21.6793  21.6793  21.6842  21.7665  21.7596  21.7596
21.6830  21.6629  21.6412  21.2018
  14 至 15 列
   21.2098  21.1814
Besttargetfunvalue =
   21.7665
nmax =
     7
Bestpopulation =
    1.7936
```

即经过16次迭代，函数最大值为21.7665。

14.5.2 函数优化求解

对于多极值点函数具有多个极值，很容易使得优化技术陷入局部最优解，求得全局优化解的概率不高，可靠性低。因此，必须建立尽可能大概率的求解全局优化解算法。

在MATLAB中，可以使用遗传算法解决标准优化算法无法解决或者很难解决的优化问题。遗传算法的搜索能力主要由选择算子及交叉算子赋存，变异算子尽可能保证算法达到全局最优，避免陷入局部最优。

在使用遗传算法求解优化的时候，经常会用到工具箱globaloptimdemos内的绘制函数图形的函数plotobjective。

【例14-7】 使用MATLAB代码编写一个待优化的目标函数，使用函数plotobjective绘制编写的函数图形，并用遗传算法对目标函数进行优化求解。

解：在MATLAB文件编辑器中编写名称为GAfcn.m的函数代码如下：

```
function f = GAfcn(x)
for j = 1:size(x,1)
    y = x(j,:);
    temp1 = 0;
    temp2 = 0;
    y1 = y(1);
    y2 = y(2);
    for i = 1:10
        temp1 = temp1 + i.*sin((i+1).*y1+i);
        temp2 = temp2 + i.*sin((i+1).*y2+i);
    end
    f(j) = temp1.*temp2;
end
```

使用plotobjective函数，编写如下MATLAB代码：

```
clear all
clc
plotobjective(@GAfcn,[-1 1;-1 1])
```

得到如图 14-3 所示的待优化的目标函数图形。

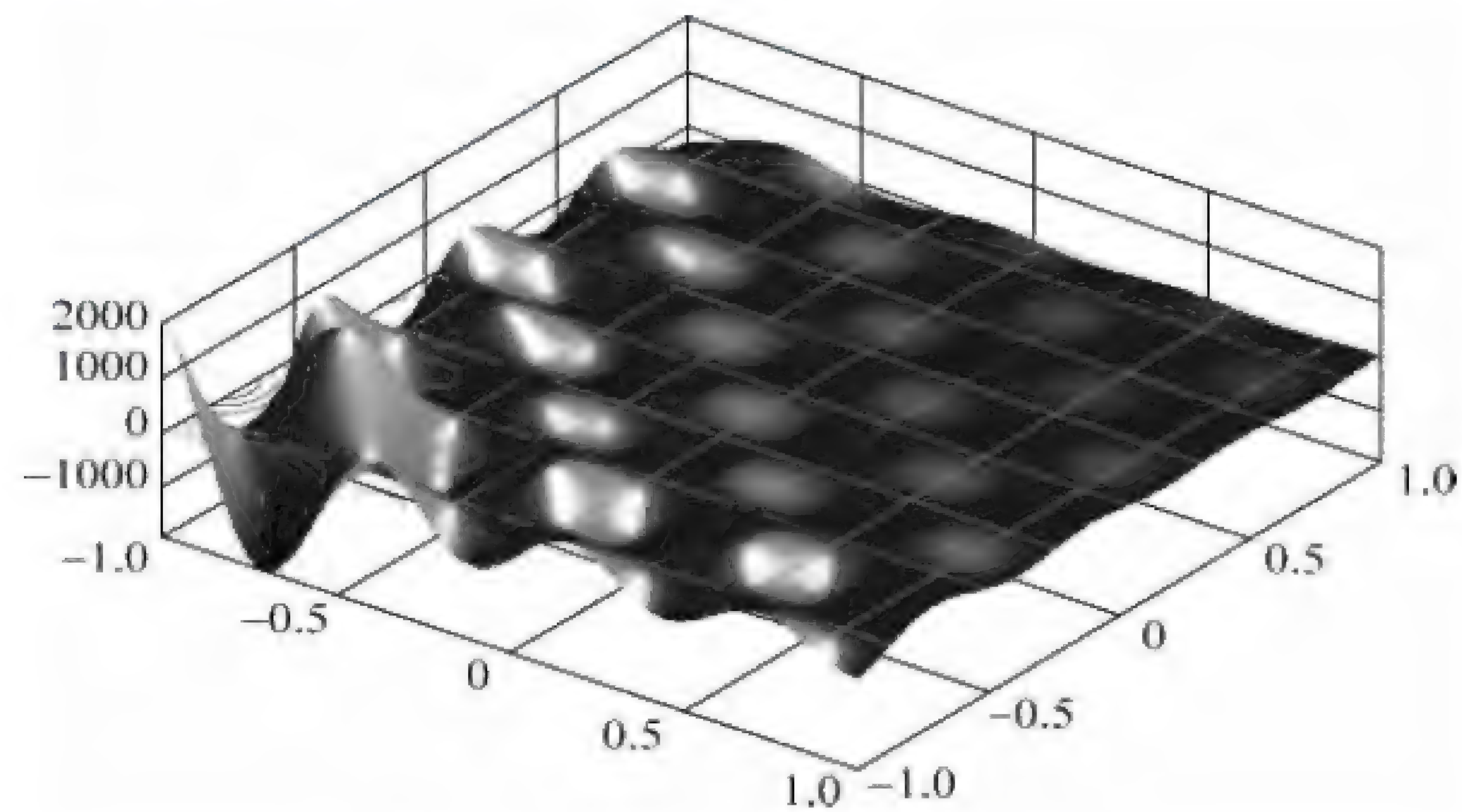


图 14-3 待优化目标函数的图形

在 MATLAB 中可以查看函数 plotobjective 的代码如下：

```
function plotobjective(fcn,range)
if(nargin == 0)
    fcn = @rastriginsfcn;
    range = [-5,5;-5,5];
end

pts = 100;
span = diff(range')/(pts - 1);
x = range(1,1): span(1) : range(1,2);
y = range(2,1): span(2) : range(2,2);

pop = zeros(pts * pts,2);
k = 1;
for i = 1:pts
    for j = 1:pts
        pop(k,:) = [x(i),y(j)];
        k = k + 1;
    end
end

values = feval(fcn,pop);
values = reshape(values,pts,pts);

surf(x,y,values)
shading interp
light
```

```
lighting phong
hold on
contour(x,y,values)
rotate3d
view(37,60)
```

使用遗传算法求解目标函数优化解时，需要设定目标函数和优化问题的变量个数。其中，GAfcn.m 作为本题的目标函数，变量个数为 3 个。利用软件自带的遗传算法函数，对目标函数进行优化求解，在 MATLAB 中编写代码如下：

```
clear all
clc
[x,fval,exitflag,output] = ga(@GAfcn,3)
```

得到结果如下：

```
Optimization terminated: average change in the fitness value less than options.
FunctionTolerance.
x =
    - 7.0026  - 13.6493   2.5293
fval =
   - 2.2183e + 03
exitflag =
      1
output =
  包含以下字段的 struct:
        problemtype: 'unconstrained'
           rngstate: [1 × 1 struct]
        generations: 118
          funccount: 5950
            message: 'Optimization terminated: average change in the fitness value less
than options.FunctionTolerance.'
      maxconstraint: []
```

本章小结

MATLAB 的优化算法包括多方面，其函数功能非常强大。利用 MATLAB 遗传算法工具箱可以快速求得函数最优解，且精度较高。

本章首先介绍了遗传算法的基本概念，然后对基本遗传算法及其 MATLAB 程序作了简单介绍，并介绍了自适应遗传算法，最后举例说明了遗传算法在求函数极值和函数优化求解中的应用。

第15章 小波变换的实现

小波分析属于时频分析的一种。传统的信号分析是建立在傅里叶(Fourier)变换的基础上的，但是傅里叶分析使用的是一种全局变换，即要么是完全在时域，要么是完全在频域，它无法表征信号的时频局域性质，但是时频局域性质恰恰是非平稳信号最根本和最关键的性质。

本章重点介绍了小波分析的基本理论、MATLAB常用小波分析函数及其应用。

学习目标：

(1) 了解小波变换原理；

(2) 掌握MATLAB中小波变换的函数；

(3) 熟练掌握小波变换的图像分解和压缩方法；

(4) 掌握小波变换在去噪和压缩中的应用。

15.1 小波变换原理

小波变换是一种信号的时间—尺度(时间—频率)分析方法，一种窗口大小固定不变形状可改变，时间窗和频率窗都可以改变的时频局部化分析方法。它具有多分辨率分析(Multi-resolution Analysis)的特点，且在时频两域都具有表征信号局部特征的能力。

小波分析方法在低频部分具有较高的频率分辨率和较低的时间分辨率，在高频部分具有较高的时间分辨率和较低的频率分辨率，所以被誉为“数学显微镜”。正是这种特性，使小波变换具有对信号的自适应性。

小波分析被看成调和分析这一数学领域半个世纪以来的工作结晶，已经广泛地应用于信号处理、图像处理、量子场论、地震勘探、语音识别与合成、音乐、雷达、CT成像、彩色复印、流体湍流、天体识别、机器视觉、机械故障诊断与监控、分形以及数字电视等科技领域。

原则上讲，传统上使用傅里叶分析的地方，都可以用小波分析取代。小波分析优于傅里叶变换的地方是在时域和频域同时具有良好的局部化性质。

设 $y(t)\in L2(\mathbf{R})L2(\mathbf{R})$表示平方可积的实数空间，即能量有限的信号空间，其傅里叶变换为 $Y(w)$。当 $Y(w)$满足允许条件

$$C_\psi = \int_{\mathbf{R}} \frac{|\hat{\psi}(\omega)|}{|\omega|}\mathrm{d}\omega < \infty$$

时，称 $y(t)$为一个基本小波或母小波（mother wavelet）。将母函数 $y(t)$经伸缩和平移后，就可以得到一个小波序列。

对于连续的情况，小波序列为

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t-b}{a}\right), \quad a,b\in\mathbf{R}; a\neq 0$$

其中，a 为伸缩因子，b 为平移因子。

对于离散的情况，小波序列为

$$\psi_{j,k}(t) = 2^{\frac{-j}{2}}\psi(2^{-j}t-k), \quad j,k\in\mathbf{Z}$$

对于任意的函数 $f(t)\in L2(\mathbf{R})$的连续小波变换为

$$W_f(a,b) = < f,\psi_{a,b} > = |a|^{-1/2}\int_{\mathbf{R}} f(t)\,\overline{\psi\left(\frac{t-b}{a}\right)}\mathrm{d}t$$

其逆变换为

$$f(t) = \frac{1}{C_\psi}\int_{\mathbf{R}^+}\int_{\mathbf{R}}\frac{1}{a^2}W_f(a,b)\psi\left(\frac{t-b}{a}\right)\mathrm{d}a\mathrm{d}b$$

小波变换的时频窗口特性与短时傅里叶的时频窗口不一样。其窗口形状为两个矩形$[b-aDy,b+aDy]$，$[(\pm w_0-DY)/a,(\pm w_0+DY)/a]$，窗口中心为$(b,\pm w_0/a)$，时窗和频窗宽分别为 aDy 和 DY/a。其中，b 仅仅影响窗口在相平面时间轴上的位置，而 a 不仅影响窗口在频率轴上的位置，也影响窗口的形状。

这样小波变换对不同的频率在时域上的取样步长是调节性的：在低频时，小波变换的时间分辨率较低，而频率分辨率较高；在高频时，小波变换的时间分辨率较高，而频率分辨率较低。这正符合低频信号变化缓慢而高频信号变化迅速的特点。

这便是它优于经典傅里叶变换与短时傅里叶变换的地方。

15.2 小波算法的 MATLAB 函数

在 MATLAB 中常用到如下小波函数：

1. waveinfo 函数

该函数可以查询小波函数的基本信息，函数使用格式如下：

```
waveinfo('wname')
```

(1) RbioNr. Nd 小波

RbioNr. Nd 函数是 reverse 双正交小波。

在 MATLAB 命令窗口，输入 waveinfo('rbio')得到该函数的主要性质：

```
>> waveinfo('rbio')
Information on reverse biorthogonal spline wavelets.

    Reverse Biorthogonal Wavelets

    General characteristics: Compactly supported
    biorthogonal spline wavelets for which
    symmetry and exact reconstruction are possible
    with FIR filters (in orthogonal case it is
    impossible except for Haar).

    Family                  Biorthogonal
    Short name              rbio
    Order Nd,Nr             Nd = 1 , Nr = 1, 3, 5
    r for reconstruction    Nd = 2 , Nr = 2, 4, 6, 8
    d for decomposition     Nd = 3 , Nr = 1, 3, 5, 7, 9
                            Nd = 4 , Nr = 4
                            Nd = 5 , Nr = 5
                            Nd = 6 , Nr = 8

    Examples                rbio3.1, rbio5.5

    Orthogonal              no
    Biorthogonal            yes
    Compact support         yes
    DWT                     possible
    CWT                     possible

    Support width           2Nd+1 for rec., 2Nr+1 for dec.
    Filters length          max(2Nd,2Nr) + 2 but essentially
    rbio Nd.Nr              lr                ld
                            effective length  effective length
                            of Hi_D           of Lo_D

    rbio 1.1                2                 2
    rbio 1.3                6                 2
    rbio 1.5                10                2
    rbio 2.2                5                 3
    rbio 2.4                9                 3
    rbio 2.6                13                3
    rbio 2.8                17                3
    rbio 3.1                4                 4
    rbio 3.3                8                 4
    rbio 3.5                12                4
    rbio 3.7                16                4
    rbio 3.9                20                4
    rbio 4.4                9                 7
    rbio 5.5                9                 11
    rbio 6.8                17                11
```

```
    Regularity for
    psi rec.              Nd - 1 and Nd - 2 at the knots
    Symmetry              yes
    Number of vanishing
    moments for psi dec.  Nd

    Remark: rbio 4.4 , 5.5 and 6.8 are such that reconstruction and
    decomposition functions and filters are close in value.

    Reference: I. Daubechies,
    Ten lectures on wavelets,
    CBMS, SIAM, 61, 1994, 271 - 280.

    See Information on biorthogonal spline wavelets.
```

(2) Gaus 小波

Gaus 小波是从高斯函数派生出来的，其表达式为

$$f(x) = C_p e^{-x^2}$$

其中，整数 p 是参数，由 p 的变化导出一系列的 $f(p)$，它满足如下条件：

$$\| f^{(p)} \|^2 = 1$$

在 MATLAB 命令窗口，输入 waveinfo('gaus')得到该函数的主要性质：

```
>> waveinfo('gaus')
Information on Gaussian wavelets.

    Gaussian Wavelets

    Definition: derivatives of the Gaussian
    probability density function.

    gaus(x,n)  = Cn * diff(exp( - x ^ 2),n) where diff denotes
    the symbolic derivative and where Cn is such that
    the 2 - norm of gaus(x,n)  = 1.

    Family              Gaussian
    Short name          gaus

    Wavelet name        'gausN' Valid choices for N are 1,2,3,...8

    Orthogonal          no
    Biorthogonal        no
    Compact support     no
    DWT                 no
    CWT                 possible

    Support width       infinite
```

```
    Effective support    [ - 5 5]
    Symmetry             yes
                         n even  ==> Symmetry
                         n odd  ==> Anti - Symmetry
```

（3）Dmey 小波

Dmey 函数可以进行快速小波变换，是 Meyer 函数的近似。

在 MATLAB 命令窗口，输入 waveinfo('dmey')得到该函数的主要性质

```
>> waveinfo('dmey')
Information on "Discrete" Meyer wavelet.

    "Discrete" Meyer Wavelet

    Definition: FIR based approximation of the Meyer Wavelet.

    Family              DMeyer
    Short name          dmey

    Orthogonal          yes
    Biorthogonal        yes
    Compact support     yes
    DWT                 possible
    CWT                 possible

    Reference: P. Abry,
    Ondelettes et turbulence,
    Diderot ed., Paris, 1997, p. 268.

    See Information on Meyer wavelet.
```

（4）Cgau 小波

Cgau 函数是从复数的高斯函数中构造出来的，它是复数形式的高斯小波，其表达式如下：

$$f(x) = C_p e^{-ix} e^{-x^2}$$

其中，参数 p 是由 p 的变化导出一系列的 $f(p)$，其满足如下条件：

$$\| f(p) \|^2 = 1$$

在 MATLAB 命令窗口，输入 waveinfo('cgau')得到该函数的主要性质：

```
>> waveinfo('cgau')
Information on complex Gaussian wavelets.

    Complex Gaussian Wavelets.

    Definition: derivatives of the complex Gaussian
```

```
    function

    cgau(x) = Cn * diff(exp( - i * x) * exp( - x ^2),n) where diff denotes
    the symbolic derivative and where Cn is a constant

    Family            Complex Gaussian
    Short name        cgau

    Wavelet name      'cgauN' Valid choices for N are 1,2,3,...8

    Orthogonal        no
    Biorthogonal      no
    Compact support   no
    DWT               no
    Complex CWT       possible

    Support width     infinite
    Symmetry          yes
                      n even ==> Symmetry
                      n odd ==> Anti - Symmetry
```

(5) Cmor 小波

Cmor 是复数形式的 morlet 小波，其表达式为

$$\psi(x) = \sqrt{\pi f_b} e^{2i\pi f_c x} e^{\frac{x}{f_b}}$$

其中，f_b是带宽参数，f_c是小波中心频率。

在 MATLAB 命令窗口，输入 waveinfo('cmor')得到该函数的主要性质：

```
>> waveinfo('cmor')
Information on complex Morlet wavelet.

    Complex Morlet Wavelet

    Definition: a complex Morlet wavelet is
        cmor(x) = (pi * Fb)^{ - 0.5} * exp(2 * i * pi * Fc * x) * exp( - (x ^2)/Fb)
    depending on two parameters:
        Fb is a bandwidth parameter
        Fc is a wavelet center frequency

    Family            Complex Morlet
    Short name        cmor

    Wavelet name      cmor"Fb" - "Fc"

    Orthogonal        no
    Biorthogonal      no
    Compact support   no
    DWT               no
    complex CWT       possible
```

```
    Support width      infinite

    Reference: A. Teolis,
    Computational signal processing with wavelets,
    Birkhauser, 1998, 65.
```

(6) Fbsp 小波

Fbsp 是复频域 B 样条小波,表达式为

$$\psi(x) = \sqrt{f_b}\left(\sin\left(\frac{f_b x}{m}\right)\right)^m e^{2i\pi f_c x}$$

其中,m 是整数型参数,f_b是带宽参数,f_c是小波中心频率。

在 MATLAB 命令窗口,输入 waveinfo('fbsp')得到该函数的主要性质:

```
>> waveinfo('fbsp')
Information on complex Frequency B-Spline wavelet.

    Complex Frequency B-Spline Wavelet

    Definition: a complex Frequency B-Spline wavelet is
        fbsp(x) = Fb^{0.5} * (sinc(Fb * x/M))^M * exp(2 * i * pi * Fc * x)
    depending on three parameters:
            M is an integer order parameter (>= 1)
            Fb is a bandwidth parameter
            Fc is a wavelet center frequency

    For M = 1, the condition Fc > Fb/2 is sufficient to ensure
    that zero is not in the frequency support interval.

    Family            Complex Frequency B-Spline
    Short name        fbsp

    Wavelet name      fbsp"M"-"Fb"-"Fc"

    Orthogonal        no
    Biorthogonal      no
    Compact support   no
    DWT               no
    complex CWT       possible

    Support width     infinite

    Reference: A. Teolis,
    Computational signal processing with wavelets,
    Birkhauser, 1998, 63.
```

(7) Shan 小波

Shan 函数是复数形式的 shannon 小波。在 B 样条频率小波中,令参数 $m=1$,就得到

Shan 小波，其表达式为

$$\psi(x) = \sqrt{f_b}\sin(f_b x)e^{2j\pi f_c x}$$

其中，f_b是带宽参数，f_c是小波中心频率。

在 MATLAB 命令窗口，输入 waveinfo('shan')得到该函数的主要性质：

```
>> waveinfo('shan')
Information on complex Shannon wavelet.

    Complex Shannon Wavelet

    Definition: a complex Shannon wavelet is
            shan(x) = Fb^{0.5} * sinc(Fb * x) * exp(2 * i * pi * Fc * x)
    depending on two parameters:
            Fb is a bandwidth parameter
            Fc is a wavelet center frequency

    The condition Fc > Fb/2 is sufficient to ensure that
    zero is not in the frequency support interval.

    Family              Complex Shannon
    Short name          shan

    Wavelet name        shan"Fb" - "Fc"

    Orthogonal          no
    Biorthogonal        no
    Compact support     no
    DWT                 no
    complex CWT         possible

    Support width       infinite

    Reference: A. Teolis,
    Computational signal processing with wavelets,
    Birkhauser, 1998, 62.
```

2. wfilters 函数

该函数是小波滤波器函数，其使用格式如下：

```
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters ('wname')
```

上面格式是用于计算正交小波或双正交小波'wname'相关的 4 个滤波器。这 4 个滤波器分别为

Lo_D—分解低通滤波器；

Hi_D—分解高通滤波器；

Lo_R—重构低通滤波器；

Hi_R—重构高通滤波器。

```
[F1,F2] = wfilters ('wname','type')
```

上面这种格式返回以下滤波器：
如果'type'='d',则返回分解滤波器 Lo_D 和 Hi_D;
如果'type'='r',则返回重构滤波器 Lo_R 和 Hi_R;
如果'type'='l',则返回低通滤波器 Lo_D 和 Lo_R;
如果'type'='h',则返回高通滤波器 Hi_D 和 Hi_R。
wfilters 函数应用程序清单：

```
%本例需要重点掌握 wfilters 函数、stem 函数的用法和底层绘图技法的属性设置
clear all;
clc
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters ('db45');
%stem 实现画出杆状图
subplot(221);
stem(Lo_D,'color','r');
xlim([0 95]);
title('分解低通滤波器','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
subplot(222);
stem(Hi_D,'color','r');
xlim([0 95]);
title('分解高通滤波器','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
subplot(223);
stem(Lo_R,'color','r');
xlim([0 95]);
title('重构低通滤波器','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
subplot(224);
stem(Hi_R,'color','r');
xlim([0 95]);
title('重构高通滤波器','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
```

运行后，得到的结果如图 15-1 所示。

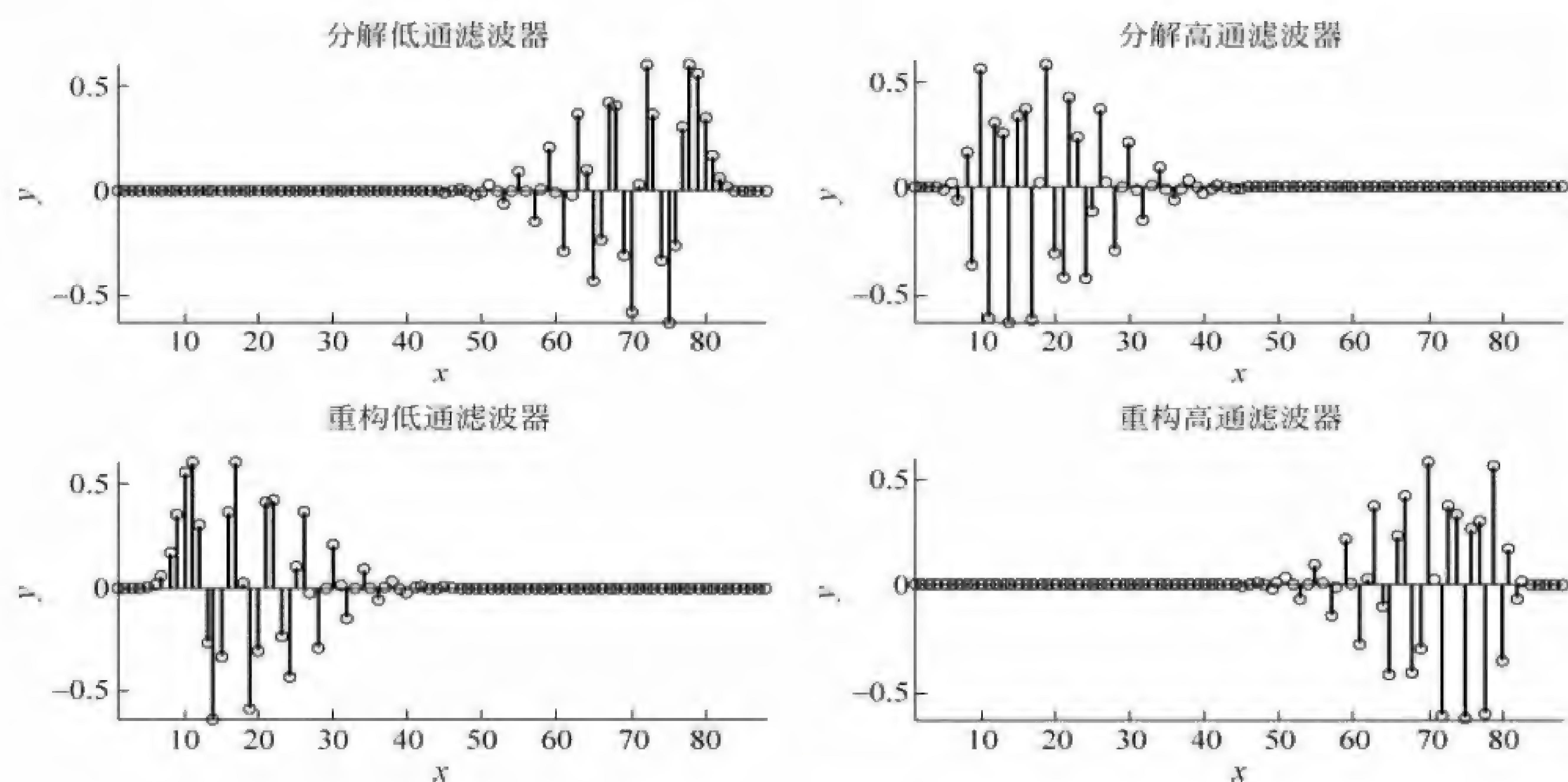


图 15-1　小波滤波器

3. dwt 函数

该函数是使用特定的小波"wname"或者特定的小波分解滤波器 Lo_D 和 Hi_D 执行单层一维小波分解。其使用格式如下：

```
[cA,cD] = dwt(X,'wname')或[cA,cD] = dwt(X, Lo_D,Hi_D)
```

下面使用 dwt 函数实现 haar 系数和 db2 系数，具体 MATLAB 代码如下：

```
clear all;
clc;
a = randn(1,256);
b = 1.5 * sin(1:256);
s = a + b;
[ca1,cd1] = dwt(s,'haar');
subplot(311);
plot(s,'k-');
title('原始信号','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
subplot(323);
plot(ca1,'k-');
title('haar 低频系数','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
subplot(324);
```

```
plot(cd1,'k-');
title('haar 高频系数','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
%计算两个相关的分解滤波器,并直接使用该滤波器计算低频和高频系数
[Lo_D,Hi_D] = wfilters('haar','d');
[ca1,cd1] = dwt(s,Lo_D,Hi_D);
%进行单尺度 db2 离散小波变换并观察最后系数的边缘效果
[ca2,cd2] = dwt(s,'db2'); %db2 也是一种小波函数
subplot(325);
plot(ca2,'k-');
title('db2 低频系数','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
subplot(326);
plot(cd2,'k-');
title('db2 高频系数','fontsize',10);
axis tight;
xlabel('x');
ylabel('y');
```

运行后,得到结果如图 15-2 所示。

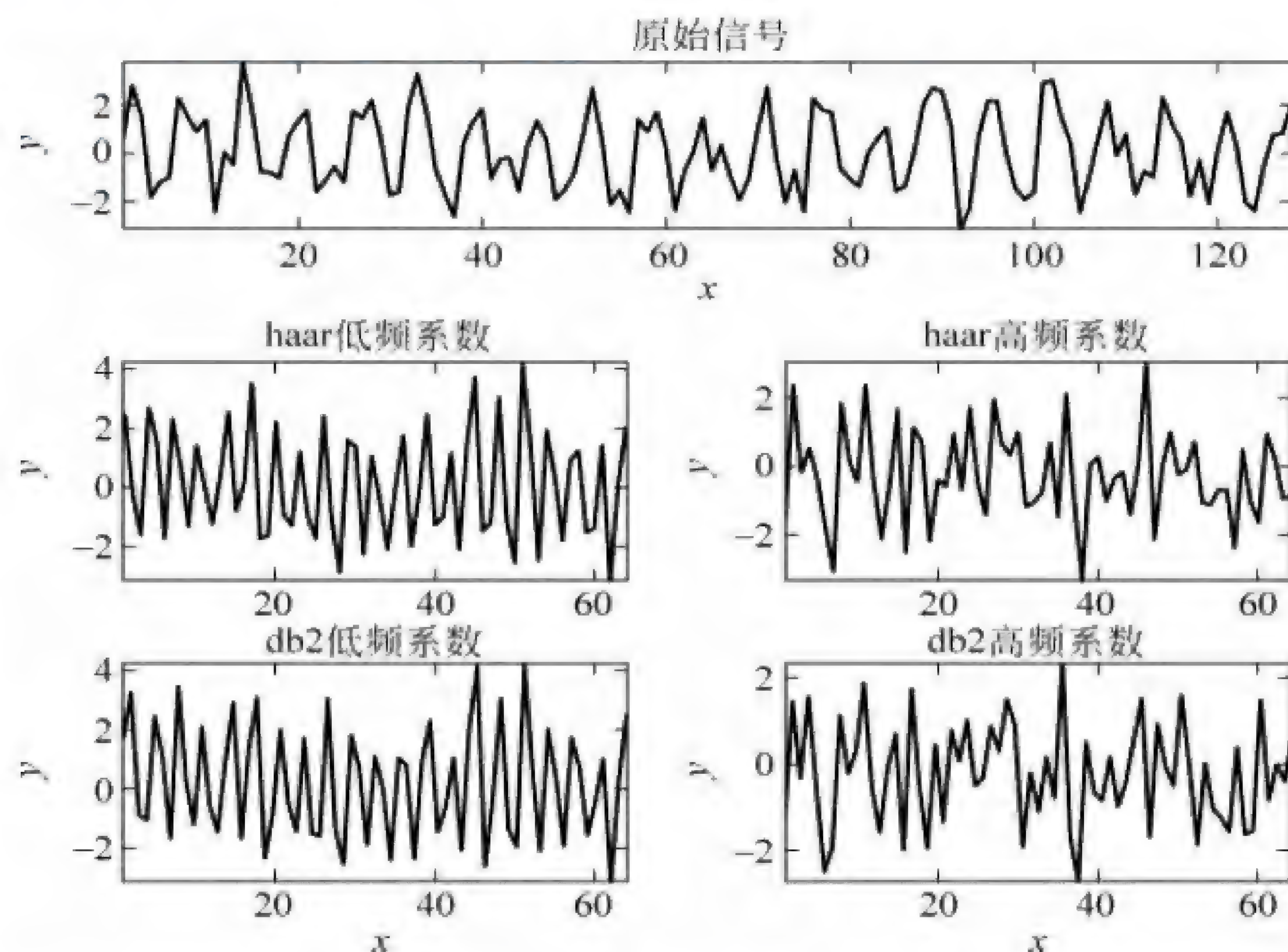


图 15-2 单层一维小波分解示意图

4. wavedec 函数

wavedec 函数使用给定的小波"wname"或者滤波器 Lo_D 和 Hi_D 进行多尺度一维小波分解。其使用格式如下:

```
[C,L] = wavedec (X,N,'wname');
```

返回信号 X 在 N 层的小波分解。N 必须是正整数。输出的结果包含分解向量 C 和相应的记录向量 L。

```
[C,L] = wavedec (X,N, Lo_D,Hi_D);
```

使用指定的低通和高通分解滤波器，返回分解结构。

5. wrcoef 函数

该函数是基于指定的小波"wname"或者重构滤波器 Lo_R 和 Hi_R，以及小波分解结构[C,L]，进行一维小波系数的单支重构。其使用格式如下：

```
X = wrcoef ('type',C,L,'wname',N);
```

基于小波分解结构[C,L]在 N 层计算重构系数向量。N 为正整数。'type'决定重构的系数是低频('type'='a')还是高频('type'='d')。

```
X = wrcoef ('type',C,L, Lo_R,Hi_R,N);
```

根据指定的重构滤波器进行系数重构。

使用 wrcoef 函数进行一维小波系数的单支重构，编写 MATLAB 代码如下：

```
clear all;
clc;
N = 80;
t = 1:N;
sig1 = sin(0.2 * t);                                          % 生成正弦信号
sig2(1:40) = ((1:40) - 1)/40;sig2(41:N) = (80 - (41:80))/40;  % 生成三角波信号
x = sig1 + sig2;
[c,l] = wavedec(x,2,'db6');                                   % 进行两层小波分解
a2 = wrcoef('a',c,l,'db6',2);
a1 = wrcoef('a',c,l,'db6',1);                                 % 重构第1~2层逼近系数
d2 = wrcoef('d',c,l,'db6',2);
d1 = wrcoef('d',c,l,'db6',1);                                 % 重构第1~2层细节系数
subplot(511);
plot(x,'linewidth',2);
ylabel('原始信号');
xlabel('信号序列');
subplot(512);
plot(a2,'linewidth',2);
ylabel('a2');
xlabel('信号序列');
subplot(513);
plot(a1,'linewidth',2);
ylabel('a1');
```

```
xlabel('信号序列');
subplot(514);
plot(d2,'linewidth',2);
ylabel('d2');
xlabel('信号序列');
subplot(515);
plot(d1,'linewidth',2);
ylabel('d1');
xlabel('信号序列');
```

程序输出的结果如图 15-3 所示。

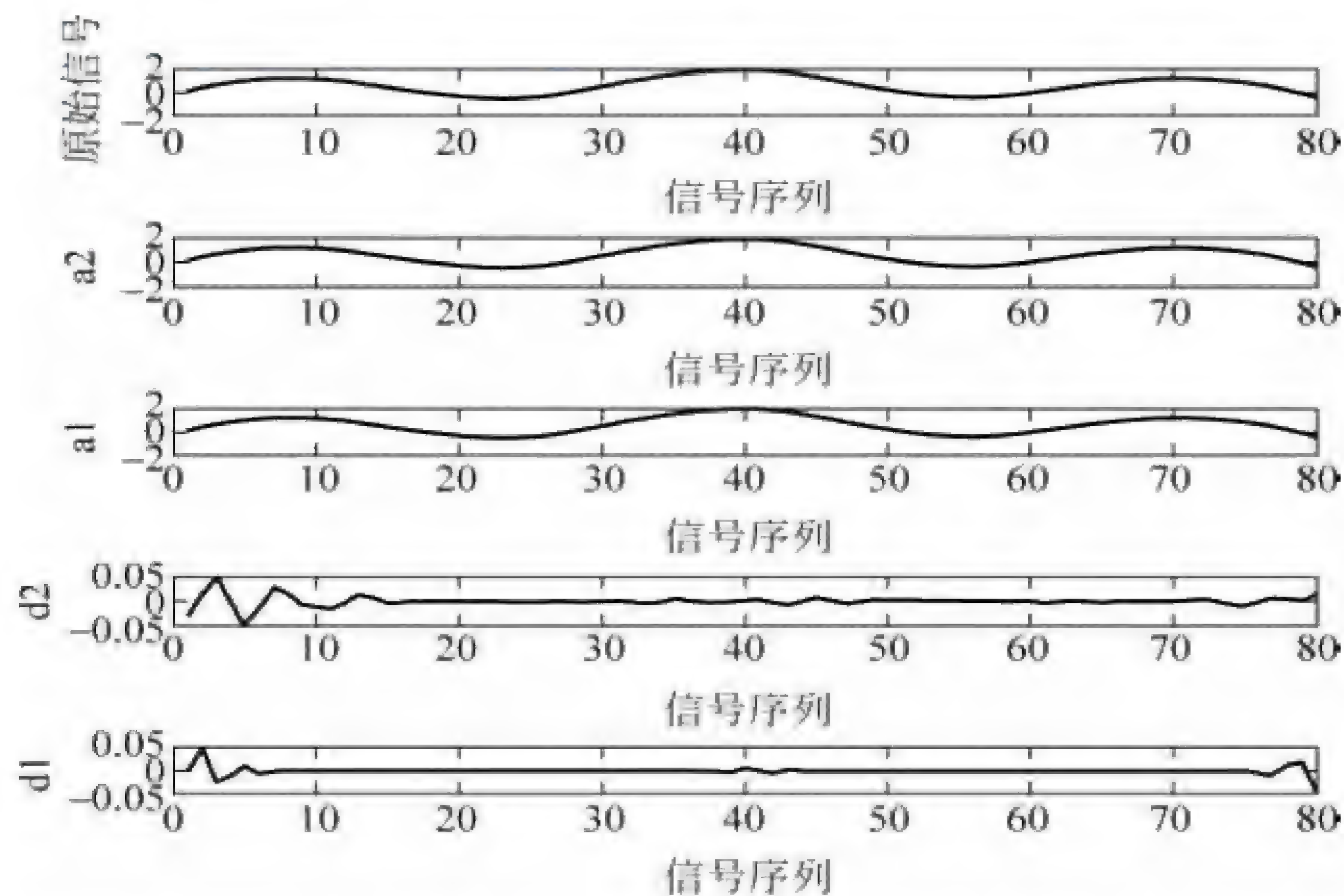


图 15-3　一维小波系数单支重构示意图

15.3　图像的分解和量化

15.3.1　一维小波变换

一维离散信号的小波分解是将信号分为低频和高频两个部分。如果是多次分解，那么就继续对上一次得到的低频部分用同样的方法进行分解，得到新的低频和高频部分，直到达到分解次数要求为止。

在分解后，系统会保留最后分解所得到的低频部分和各次分解所得到的高频部分，来实现图像的重建。

MATLAB 就是通过函数 dwt 和 idwt 实现一维信号的分解与重建。

【例 15-1】 构建长度为 512 的一维原始信号，用函数 idwt 实现该一维信号的分解与重建。

解：根据函数的使用方法，编写如下代码：

```
clear all
clc
N = 512;
S = randn(1,N);                        % 构建长度为 512 的原始信号 S
[C1,L1] = dwt(S,'db1');                % 对信号进行第一次分解
[C2,L2] = dwt(C1,'db1');               % 对信号进行第二次分解
[C3,L3] = dwt(C2,'db1');               % 对信号进行第三次分解
c2 = idwt(C3,L3,'db1');                % 重建得到第二次分解时的低频部分
c1 = idwt(c2,L2,'db1');                % 重建得到第一次分解时的低频部分
s = idwt(c1,L1,'db1');                 % 重建原始信号
subplot(6,1,1);
plot(S);
title('原始信号 S');
ylabel('S');
xlabel('信号序列');
subplot(6,1,2);
plot(C3);
title('三次分解后的低频部分');
ylabel('C3');
xlabel('信号序列');
subplot(6,1,3);
plot(L1);
title('一次分解后的高频部分');
ylabel('L1');
xlabel('信号序列');
subplot(6,1,4);
plot(L2);
title('二次分解后的高频部分');
ylabel('L2');
xlabel('信号序列');
subplot(6,1,5);
plot(L3);
title('三次分解后的高频部分');
ylabel('L3');
xlabel('信号序列');
subplot(6,1,6);
plot(s);
title('重建信号 S');
ylabel('s');
xlabel('信号序列');
```

运行后，得到结果如图 15-4 所示。

通过图 15-4 中重构后的信号与原信号相比，可以看出误差在实验许可范围内，小波分解是可行的。

15.3.2 二维变换体系

假定二维尺度函数可分离，则有

$$\varphi(x,y) = \varphi(x)\varphi(y)$$

其中 $\varphi(x)$、$\varphi(y)$是两个一维尺度函数。若 $\psi(x)$是相应的小波，那么下列三个二维基本

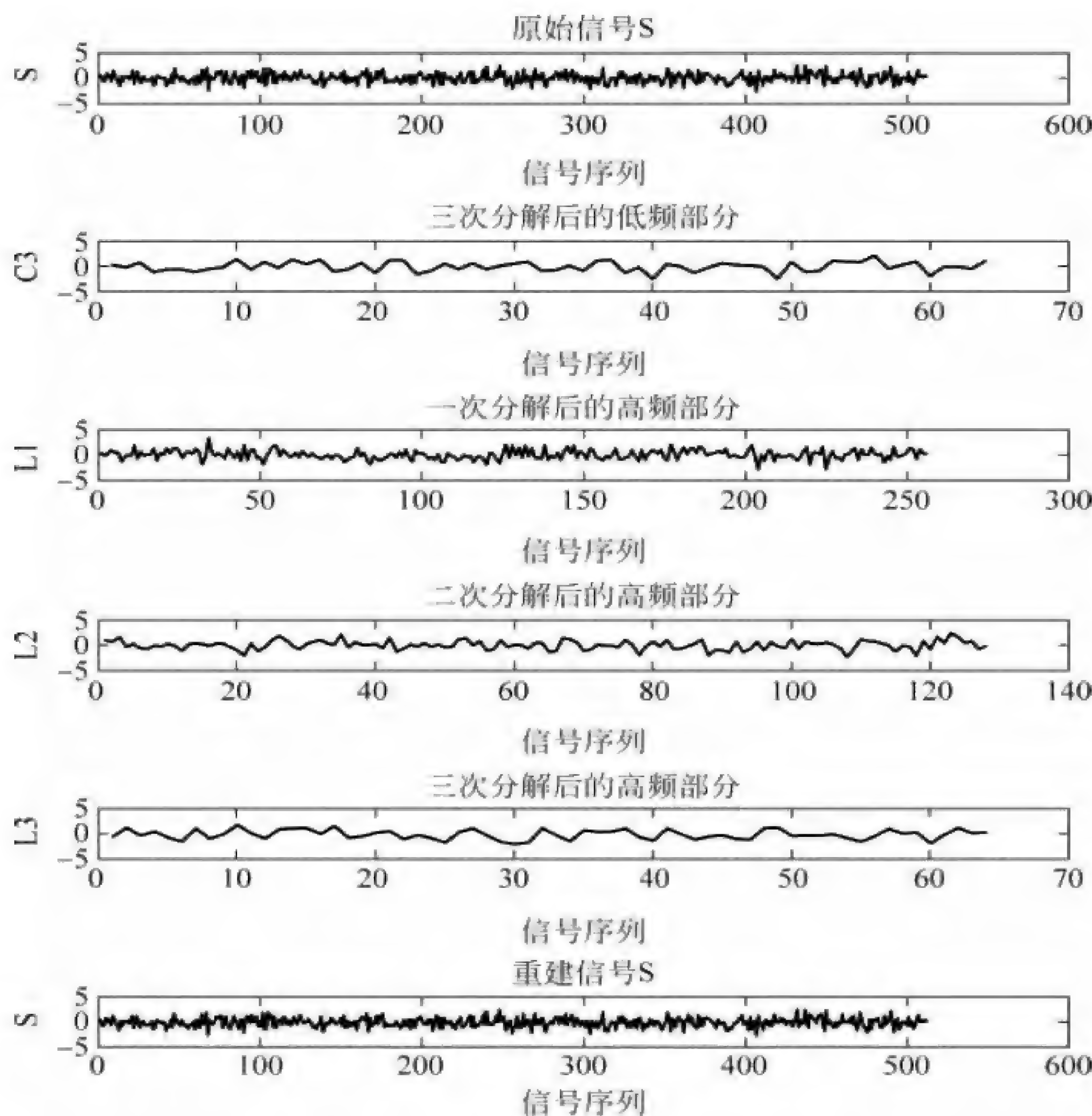


图 15-4　一维信号的分解与重建结果

小波

$$\psi^1_{(x,y)} = \varphi(x)\psi(y)$$
$$\psi^2_{(x,y)} = \psi(x)\varphi(y)$$
$$\psi^3_{(x,y)} = \psi(x)\psi(y)$$

与 $\varphi(x,y)$ 建立了二维小波变换的基础。

【例 15-2】 使用 dwt 函数，对图 15-5 实现二维小波变换。

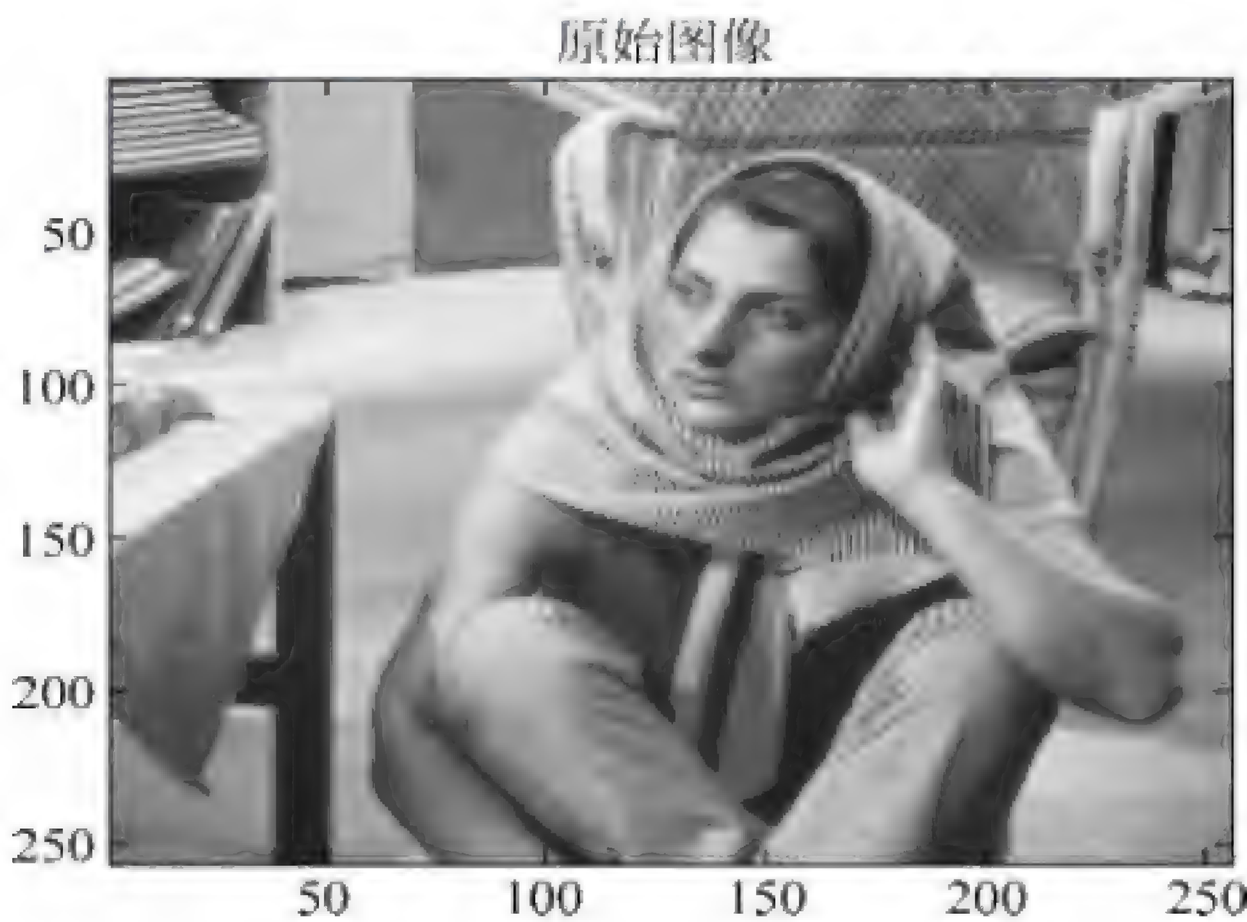


图 15-5　原始图像

解：根据 dwt 函数使用格式和二维小波变换，编写以下 MATLAB 代码：

```
clear all;
clc;
T = 256;                      % 图像维数
SUB_T = T/2;                  % 子图维数
% 调原始图像矩阵
load wbarb;                   % 下载图像
f = X;                        % 原始图像
% 进行二维小波分解
l = wfilters('db10','l');     % db10(消失矩为 10)低通分解滤波器冲击响应(长度为 20)
L = T - length(l);
l_zeros = [l,zeros(1,L)];     % 矩阵行数与输入图像一致,为 2 的整数幂
h = wfilters('db10','h');     % db10(消失矩为 10)高通分解滤波器冲击响应(长度为 20)
h_zeros = [h,zeros(1,L)];     % 矩阵行数与输入图像一致,为 2 的整数幂
for i = 1:T;                  % 列变换
    row(1:SUB_T,i) = dyaddown( ifft( fft(l_zeros). * fft(f(:,i)') ) ).';   % 圆周卷积
                                                                            <->FFT
    row(SUB_T+1:T,i) = dyaddown( ifft( fft(h_zeros). * fft(f(:,i)') ) ).';   % 圆周卷积
                                                                              <->FFT
end;
for j = 1:T;                  % 行变换
    line(j,1:SUB_T) = dyaddown( ifft( fft(l_zeros). * fft(row(j,:)) ) );   % 圆周卷积
                                                                          <->FFT
    line(j,SUB_T+1:T) = dyaddown( ifft( fft(h_zeros). * fft(row(j,:)) ) );   % 圆周卷积
                                                                            <->FFT
end;
decompose_pic = line;         % 分解矩阵
% 图像分为四块
lt_pic = decompose_pic(1:SUB_T,1:SUB_T);          % 在矩阵左上方为低频分量——fi(x) * fi(y)
rt_pic = decompose_pic(1:SUB_T,SUB_T+1:T);        % 矩阵右上为——fi(x) * psi(y)
lb_pic = decompose_pic(SUB_T+1:T,1:SUB_T);        % 矩阵左下为——psi(x) * fi(y)
rb_pic = decompose_pic(SUB_T+1:T,SUB_T+1:T); % 右下方为高频分量——psi(x) * psi(y)
% 分解结果显示
figure(1);
colormap(map);
subplot(2,1,1);
image(f);
title('原始图像');
subplot(2,1,2);
image(abs(decompose_pic));
title('分解后图像');
figure(2);
colormap(map);
subplot(2,1,1);
image(abs(lt_pic));                               % 左上方为低频分量——fi(x) * fi(y)
title('低频分量');
subplot(2,1,2);
image(abs(rb_pic));
```

```
title('高频分量');
%重构源图像及结果显示
l_re = l_zeros(end: - 1:1);                          %重构低通滤波
l_r = circshift(l_re',1)';                           %位置调整
h_re = h_zeros(end: - 1:1);                          %重构高通滤波
h_r = circshift(h_re',1)';                           %位置调整

top_pic = [lt_pic,rt_pic];                           %图像上半部分
t = 0;
for i = 1:T;                                         %行插值低频

    if (mod(i,2) == 0)
        topll(i,:) = top_pic(t,:);                   %偶数行保持
    else
        t = t + 1;
        topll(i,:) = zeros(1,T);                     %奇数行为零
    end
end;
for i = 1:T;                                         %列变换
    topcl_re(:,i) = ifft( fft(l_r). * fft(topll(:,i)') )';    %圆周卷积<->FFT
end;

bottom_pic = [lb_pic,rb_pic];                        %图像下半部分
t = 0;
for i = 1:T;                                         %行插值高频
    if (mod(i,2) == 0)
        bottomlh(i,:) = bottom_pic(t,:);             %偶数行保持
    else
        bottomlh(i,:) = zeros(1,T);                  %奇数行为零
        t = t + 1;
    end
end;
for i = 1:T;                                         %列变换
    bottomch_re(:,i) = ifft( fft(h_r). * fft(bottomlh(:,i)') )';  %圆周卷积<->FFT
end;

construct1 = bottomch_re + topcl_re;                 %列变换重构完毕

left_pic = construct1(:,1:SUB_T);                    %图像左半部分
t = 0;
for i = 1:T;                                         %列插值低频

    if (mod(i,2) == 0)
        leftll(:,i) = left_pic(:,t);                 %偶数列保持
    else
        t = t + 1;
        leftll(:,i) = zeros(T,1);                    %奇数列为零
    end
end;
for i = 1:T;                                         %行变换
    leftcl_re(i,:) = ifft( fft(l_r). * fft(leftll(i,:)) );    %圆周卷积<->FFT
end;
```

```
right_pic = construct1(:,SUB_T + 1:T);                          % 图像右半部分
t = 0;
for i = 1:T;                                                    % 列插值高频
    if (mod(i,2) == 0)
        rightlh(:,i) = right_pic(:,t);                          % 偶数列保持
    else
        rightlh(:,i) = zeros(T,1);                              % 奇数列为零
        t = t + 1;
    end
end;
for i = 1:T;                                                    % 行变换
    rightch_re(i,:) = ifft( fft(h_r). * fft(rightlh(i,:)) );   % 圆周卷积<->FFT
end;

construct_pic = rightch_re + leftcl_re;                         % 重建全部图像

%结果显示
figure(3);
colormap(map);
subplot(2,1,1);
image(f);
title('源图像显示');
subplot(2,1,2);
image(abs(construct_pic));
title('重构源图像显示');
error = abs(construct_pic - f);
figure(4);
mesh(error);                                                    % 误差三维图像
title('重构图形与原始图像误值');
```

运行后，得到分解前后图像对比如图 15-6 所示。

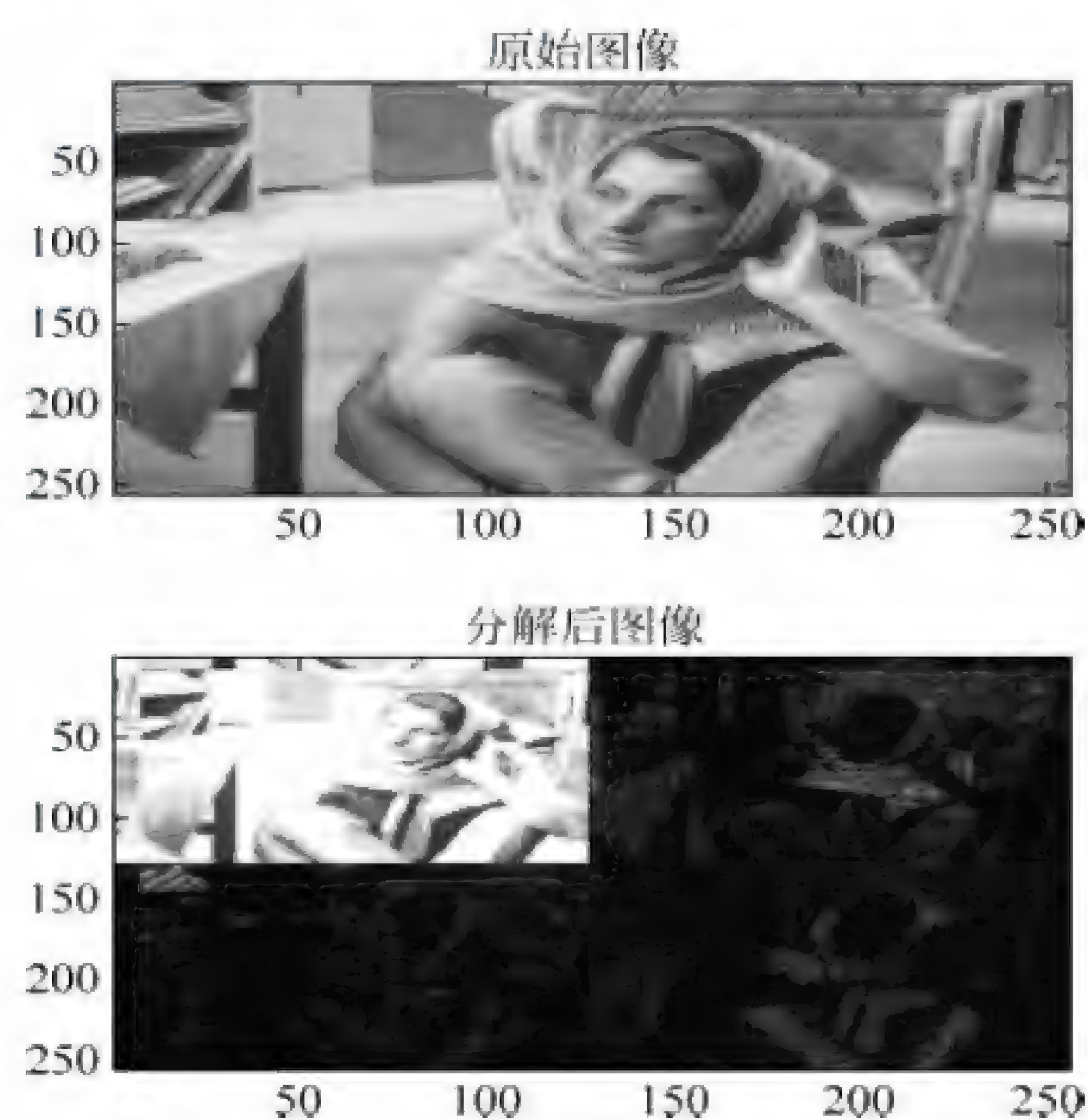


图 15-6　分解前后图像对比

15.4 小波变换经典案例

15.4.1 去噪

噪声一般理解为妨碍人的视觉器官对所接受声音、图形或图像源进行理解或分析的各种因素。噪声对声音、图形或图像处理十分重要，影响声音、图形或图像的输入、采集、处理的各个环节。

如果噪声不能很好地抑制，必然影响处理的全过程和输出结果。因此有目的地从测量数据中获取有效信息，去除噪声，成为许多分析过程中的一个重要环节。

【例 15-3】 利用小波分析对有噪声的信号进行去噪处理以恢复其原始信号。

解：在 MATLAB 命令窗口输入以下程序：

```
clear all
clc
load leleccum;                      %装载采集的信号
s = leleccum(1:2000);               %将信号中第 1 到第 1000 个采样点赋给 s
ls = length(s);
subplot(2,2,1);
plot(s);
title('原始信号图形');
grid on
%用 db1 小波对原始信号进行 3 层分解并提取系数
[c,l] = wavedec(s,3,'db1');
ca3 = appcoef(c,l,'db1',3);
cd3 = detcoef(c,l,3);
cd2 = detcoef(c,l,2);
cd1 = detcoef(c,l,1);
%对信号进行强制性去噪处理并图示结果
cdd3 = zeros(1,length(cd3));
cdd2 = zeros(1,length(cd2));
cdd1 = zeros(1,length(cd1));
c1 = [ca3 cdd3 cdd2 cdd1];
s1 = waverec(c1,l,'db1');
subplot(2,2,3);
plot(s1);
title('强制去噪后的信号');
grid on
%用默认阈值对信号进行去噪处理并图示结果
%用 ddencmp()函数获得信号的默认阈值,使用 wdencmp()命令函数实现去噪过程
[thr,sorh,keepapp] = ddencmp('den','wv',s);
s2 = wdencmp('gbl',c,l,'db1',3,thr,sorh,keepapp);
subplot(2,2,2);
plot(s2);
title('默认阈值去噪后的信号');
grid on
```

```
%用给定的软阈值进行去噪处理
cd1soft = wthresh(cd1,'s',2.65);
cd2soft = wthresh(cd2,'s',1.53);
cd3soft = wthresh(cd3,'s',1.76);
c2 = [ca3 cd3soft cd2soft cd1soft];
s3 = waverec(c2,l,'db1');
subplot(2,2,4);
plot(s3);
title('给定软阈值去噪后的信号');
grid on
```

运行后，得到信号去噪结果如图 15-7 所示。

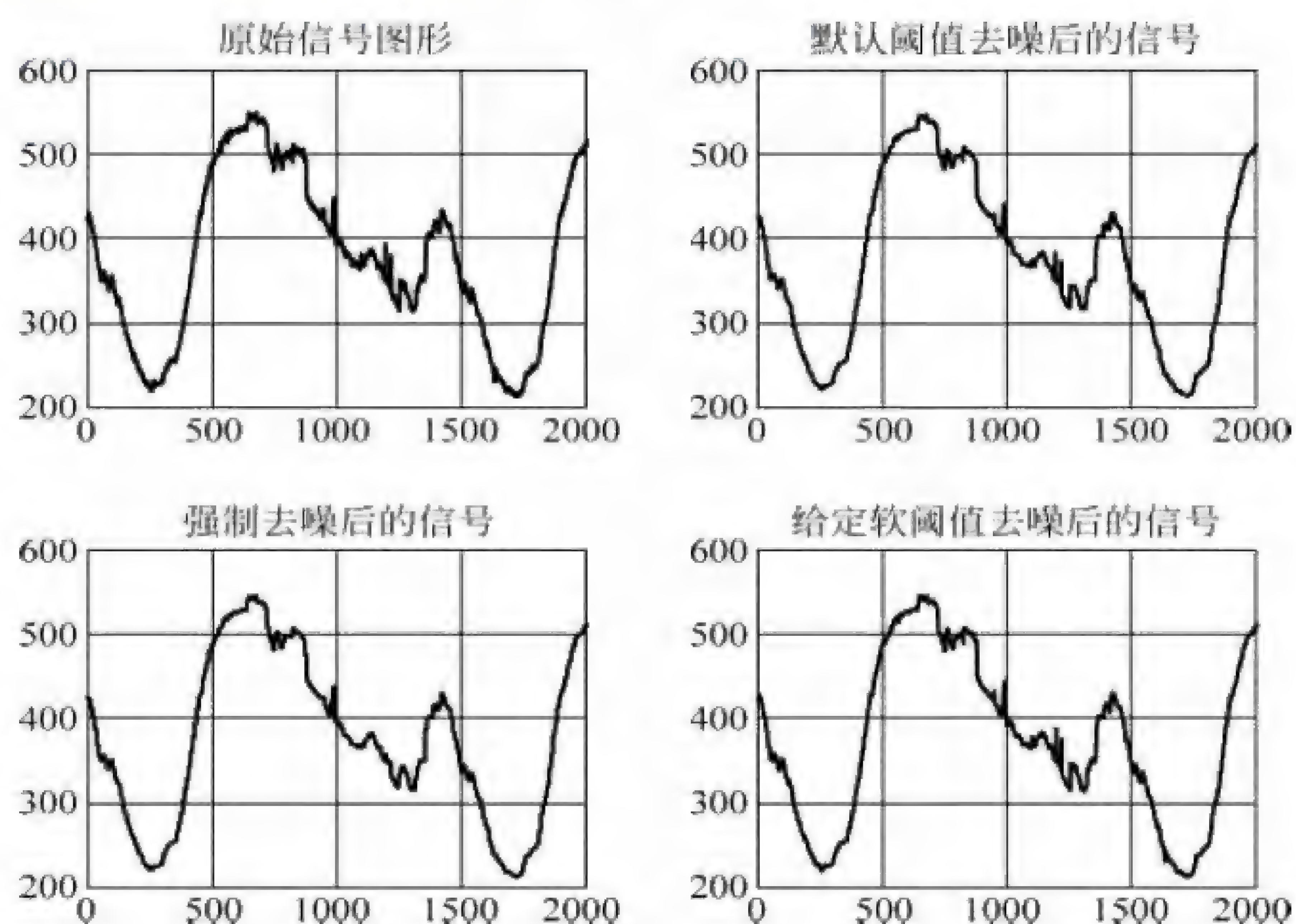


图 15-7　信号去噪结果

目前，默认阈值去噪和给定软阈值去噪这两种处理方法在实际中应用更为广泛一些。从图 15-7 可以看出，应用强制去噪处理后的信号较为光滑，但是很可能丢失了信号中的一些有用成分。

【例 15-4】 利用小波分析对含噪正弦波进行去噪。

解：编写 MATLAB 代码如下：

```
clear all
clc
N = 150;
t = 1:N;
x = sin(0.4 * t);
%加噪声
load noissin;
ns = noissin;
```

```
%显示波形
subplot(3,1,1);
plot(t,x);
title('原始正弦信号');
subplot(3,1,2);
plot(ns);
title('含噪正弦波');
%小波去噪
xd = wden(ns,'minimaxi','s','one',4,'db3');
subplot(3,1,3);
plot(xd);
title('去噪后的正波形信号');
```

运行后,得到结果如图 15-8 所示。

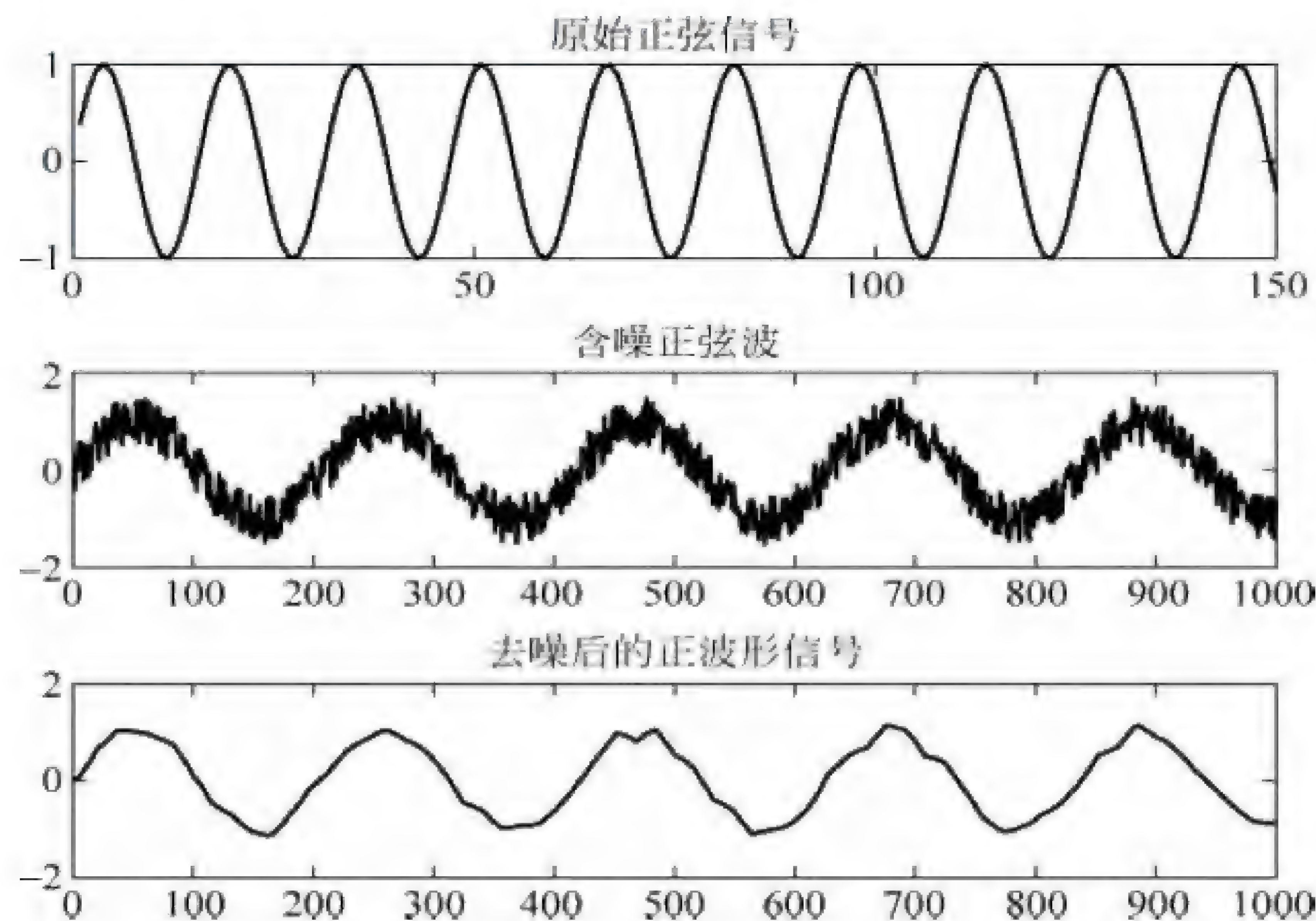


图 15-8　含噪正弦波去噪结果

去噪后的信号基本上恢复了原始信号的形状,并明显地除去了噪声所引起的干扰,这在图 15-8 中可以看出。但恢复后的信号和原始信号相比,有明显的改变。这是因为在进行去噪处理的过程中,所用的分析小波和细节系数阈值不恰当所致。

15.4.2　压缩

图像或声音信号压缩在图像或声音的传输和储存中起着至关重要的作用。小波变换由于具有良好的时域局部化性能,有效地克服了傅里叶变换在处理非平稳的复杂图像或声音信号时的局限性,因而在图像或声音压缩领域得到了广泛应用。

【例 15-5】　利用小波分析对给定信号进行压缩处理。

解: 使用函数 wdcbm()获取信号压缩阈值,然后采用函数 wdencmp()实现信号压缩。

```
clear all
clc
load nelec;                         % 装载信号
index = 1:256;
x = nelec(index);
[c,l] = wavedec(x,6,'haar');        % 用小波 haar 对信号进行 6 层分解
alpha = 1.3;
[thr,nkeep] = wdcbm(c,l,alpha);  % 获取信号压缩的阈值
[xd,cxd,lxd,perf0,perfl2] = wdencmp('lvd',c,l,'haar',6,thr,'s');
% 对信号进行压缩
subplot(2,1,1);
plot(index,x);
title('初始信号');
subplot(2,1,2);
plot(index,xd);
title('经过压缩处理的信号');
```

运行后,得到结果如图 15-9 所示。

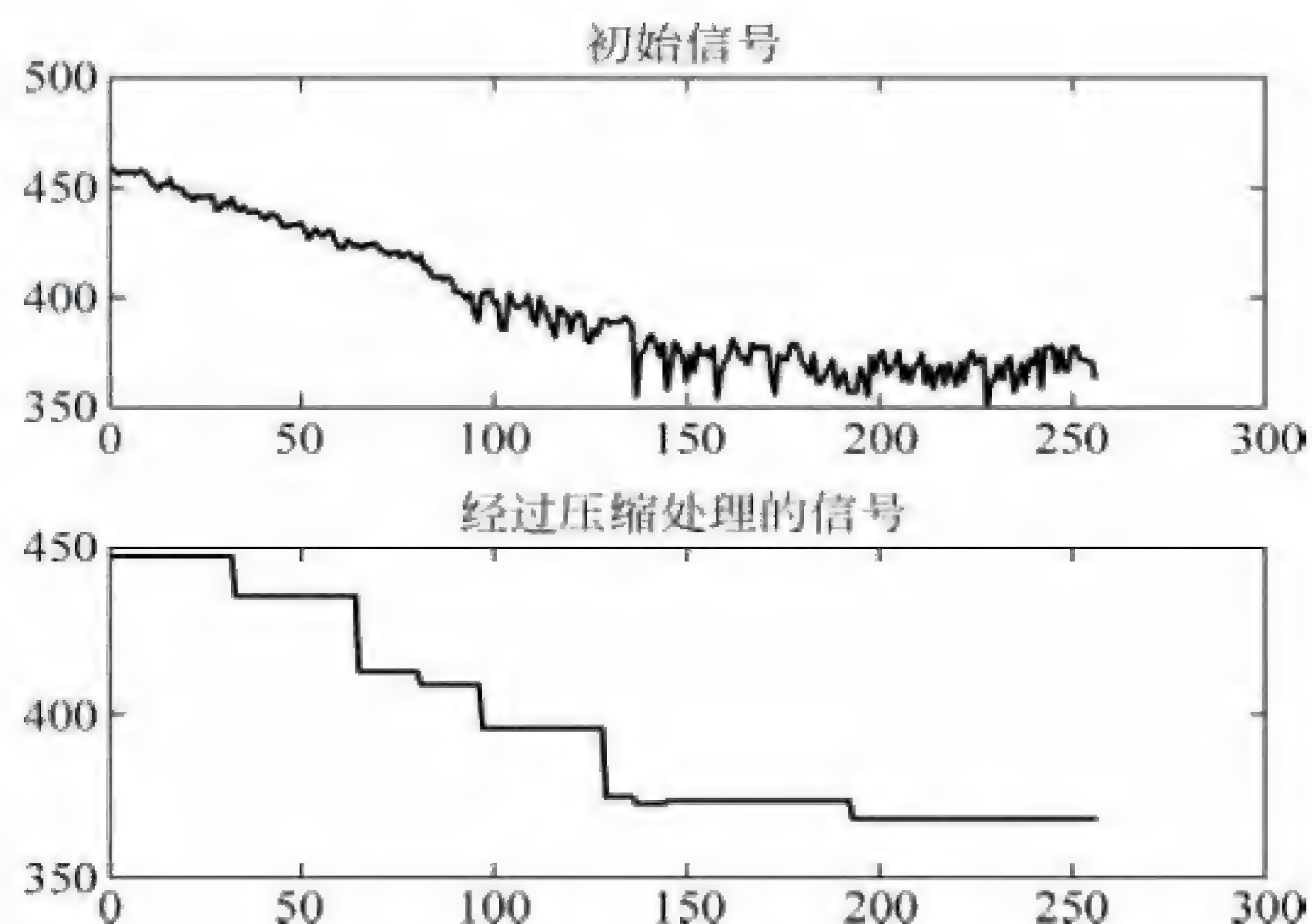


图 15-9　压缩处理前后结果比较图

【例 15-6】　利用小波分析对图 15-5 进行局部压缩。

解:根据小波算法,编写 MATLAB 代码如下:

```
clear all
clc
load wbarb
% 使用 sym4 小波对信号进行一层小波分解
[ca1,ch1,cv1,cd1] = dwt2(X,'sym4');
codca1 = wcodemat(ca1,192);
codch1 = wcodemat(ch1,192);
codcv1 = wcodemat(cv1,192);
codcd1 = wcodemat(cd1,192);
% 将四个系数图像组合为一个图像
codx = [codca1,codch1,codcv1,codcd1];
```

```
%复制原图像的小波系数
rca1 = ca1;
rch1 = ch1;
rcv1 = cv1;
rcd1 = cd1;
%将三个细节系数的中部置零
rch1(33:97,33:97) = zeros(65,65);
rcv1(33:97,33:97) = zeros(65,65);
rcd1(33:97,33:97) = zeros(65,65);
codrca1 = wcodemat(rca1,192);
codrch1 = wcodemat(rch1,192);
codrcv1 = wcodemat(rcv1,192);
codrcd1 = wcodemat(rcd1,192);
%将处理后的系数图像组合为一个图像
codrx = [codrca1,codrch1,codrcv1,codrcd1];
%重建处理后的系数
rx = idwt2(rca1,rch1,rcv1,rcd1,'sym4');
subplot(221);
image(wcodemat(X,192)),colormap(map);
title('原始图像');
subplot(222);
image(codx),colormap(map);title('一层分解后各层系数图像');
subplot(223);
image(wcodemat(rx,192)),colormap(map);
title('压缩图像');
subplot(224);
image(codrx),colormap(map);
title('处理后各层系数图像');
%求压缩信号的能量成分
per = norm(rx)/norm(X)
%求压缩信号与原信号的标准差
err = norm(rx - X)
```

运行后得到结果如图 15-10 所示。

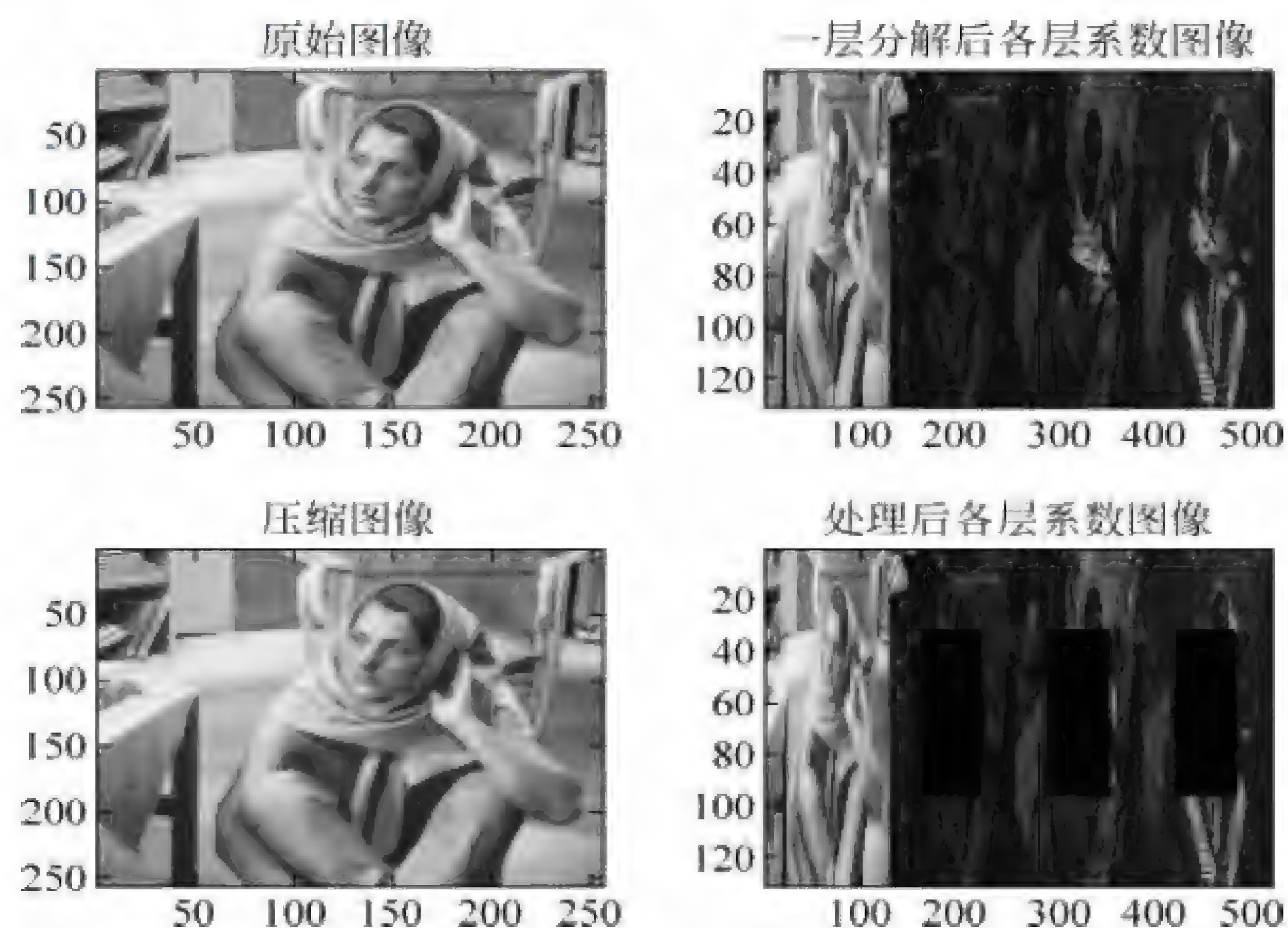


图 15-10　小波变换实现的局部压缩图像

编写的 MATLAB 代码中，把图像中部的细节系数都置零，从压缩图像中可以很明显地看出只有中间部分变得模糊，而其他部分的细节信息仍然可以分辨清楚。

本章小结

小波分析在信号处理中有很多的应用，本章首先介绍了小波变换的原理及 MATLAB 函数的使用，然后重点介绍了小波分析对图像的分解和量化，并对小波算法在去噪和压缩方面的应用做了举例说明。

第16章 神经网络的实现

人工神经网络(artificial neural networks,ANN)也简称为神经网络(NN)。ANN是一种模仿动物神经网络行为特征,进行分布式并行信息处理的优化算法模型。该算法依靠系统的复杂程度,通过调整内部大量节点之间相互连接的关系,从而达到处理信息的目的。

随着MATLAB和神经网络的发展,MATLAB神经网络工具箱提供的神经网络函数越来越多,极大提高了神经网络算法的应用。

学习目标:

(1) 了解神经网络基本概念;

(2) 掌握MATLAB神经网络工具箱;

(3) 熟练运行MATLAB实现神经网络算法。

16.1 人工神经网络基本概念

人工神经网络的构筑理念是受到生物(人或其他动物)神经网络功能的运作启发而产生的。人工神经网络通常是通过一个基于数学统计学类型的学习方法(learning method)得以优化,所以人工神经网络也是数学统计学方法的一种实际应用。

通过统计学的标准数学方法,我们能够得到大量可以用函数来表达的局部结构空间,另一方面在人工智能学的人工感知领域,我们通过数学统计学来做人工感知方面的决定问题,这种方法比起正式的逻辑学推理演算更具有优势。

人工神经网络模型主要考虑网络连接的拓扑结构、神经元的特征和学习规则等。目前,已有近40种神经网络模型,其中有反传网络、感知器、自组织映射、Hopfield网络、玻耳兹曼机、适应谐振理论等。根据连接的拓扑结构,神经网络模型可以分为

(1) 前向网络:网络中各个神经元接受前一级的输入,并输出到下一级,网络中没有反馈,可以用一个有向无环路图表示。这种网络实现信号从输入空间到输出空间的变换,它的信息处理能力来自于简单非线性函数的多次复合。前向网络结构简单,易于实现。反传网络是一种典型的前向网络。

（2）反馈网络：网络内神经元间有反馈，可以用一个无向完备图表示。这种神经网络的信息处理是状态的变换，可以用动力学系统理论处理。系统的稳定性与联想记忆功能有密切关系。Hopfield 网络和玻耳兹曼机均属于这种类型。

学习是神经网络研究的一个重要内容，神经网络的适应性是通过学习实现的，根据环境的变化，对权值进行调整，改善系统的行为。

由 Hebb 提出的 Hebb 学习规则为神经网络的学习算法奠定了基础。Hebb 规则认为学习过程最终发生在神经元之间的突触部位，突触的联系强度随着突触前后神经元的活动而变化。

在此基础上，人们提出了各种学习规则和算法，以适应不同网络模型的需要。有效的学习算法，使神经网络能够通过连接权值的调整，构造客观世界的内在表示，形成具有特色的信息处理方法，信息存储和处理体现在网络的连接中。

16.2 MATLAB 神经网络工具箱

MATLAB 包含进行神经网络应用设计和分析的许多工具箱函数。初学者可以利用该工具箱来深刻理解各种算法的内在实质。对研究者而言，该工具箱强大的扩充功能将令其工作游刃有余。最关键的是 MATLAB 丰富的函数可以节约大量的编程时间。

MATLAB 神经网络工具箱函数如表 16-1 所示。

表 16-1　神经网络工具函数

	名称	用　　途	名称	用　　途
创建函数	newp	创建感知器网络	newlind	设计一线性层
	newlin	创建一线性层	newff	创建一前馈 BP 网络
	newcf	创建一多层前馈 BP 网络	newfftd	创建一前馈输入延迟 BP 网络
	newrb	设计一径向基网络	newrbe	设计一严格的径向基网络
	newgrnn	设计一广义回归神经网络	newpnn	设计一概率神经网络
	newc	创建一竞争层	newsom	创建一自组织特征映射
	newhop	创建一 Hopfield 递归网络	newelm	创建一 Elman 递归网络
应用函数	sim	仿真一个神经网络	init	初始化一个神经网络
	adapt	神经网络的自适应化	train	训练一个神经网络
学习函数	learnp	感知器学习函数	learnpn	标准感知器学习函数
	learnwh	Widrow_Hof f 学习规则	learngd	BP 学习规则
	learngdm	带动量项的 BP 学习规则	learnk	Kohonen 权学习函数
	learncon	Conscience 阈值学习函数	learnsom	自组织映射权学习函数
训练函数	trainwb	网络权与阈值的训练函数	traingdm	梯度下降 w/动量的 BP 算法训练函数
	traingd	梯度下降的 BP 算法训练函数	traingda	梯度下降 w/自适应 lr 的 BP 算法训练函数
	traingdx	梯度下降 w/动量和自适应 lr 的 BP 算法训练函数	trainlm	Levenberg_ Marquardt 的 BP 算法训练函数
绘图函数	plotes	绘制误差曲面	plotsom	绘制自组织映射图
	plotep	绘制权和阈值在误差曲面上的位置		

16.2.1 常用神经元激活函数

本节介绍几种常用的 MATLAB 神经元激活函数，具体如下：

1. hardlim 函数

该函数是限制函数，其输出范围是{0,1}，函数调用格式如下：

```
A = hardlim(N,FP)
```

其中，N 为输入向量，FP 为功能参数(可省略)。

hardlim 函数的 MATLAB 应用示例如下：

```
clear all
clc
a = -2:0.05:2;
b = hardlim(n);
plot(a,b)
grid on
```

运行后，得到如图 16-1 所示的 hardlim 函数图形。

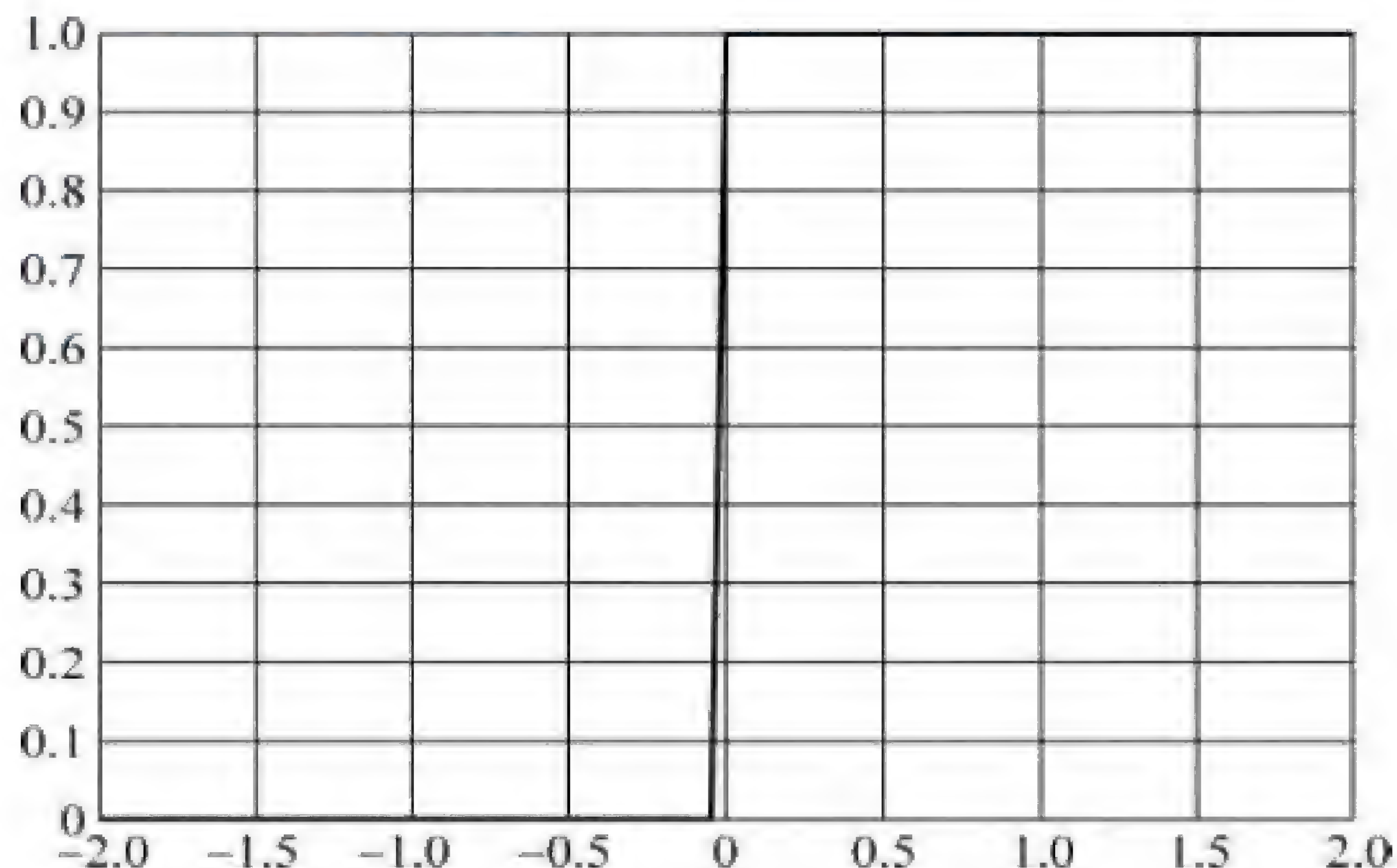


图 16-1　hardlim 函数图形

2. hardlims 函数

该函数为二值函数，其输出值为{-1,1}，调用格式如下：

```
A = hardlims(N,FP)
```

其中，N 为输入向量，FP 为功能参数(可省略)。

函数的 MATLAB 应用示例如下：

```
clear all
clc
a = -2:0.05:2;
b = hardlims(a);
plot(a,b)
grid on
```

运行后，得到如图 16-2 所示的 hardlims 函数图形。



图 16-2 hardlims 函数图形

如果需要在一个网络中使用该函数作为传递函数，可以使用如下程序：

```
net.layers{i}.transferFcn = 'hardlims';
```

3. purelin 函数

该函数是线性函数，斜率为 1，其调用格式如下：

```
A = purelin(N,FP)
```

其中，N 为输入向量，FP 为功能参数(可省略)。

函数的 MATLAB 应用示例如下：

```
clear all
clc
a = -2:0.05:2;
b = purelin(a);
plot(a,b)
grid on
```

运行后，得到如图 16-3 所示的 purelin 函数图形。

4. logsig 函数

该函数调用格式如下：

```
A = logsig (N,FP)
```

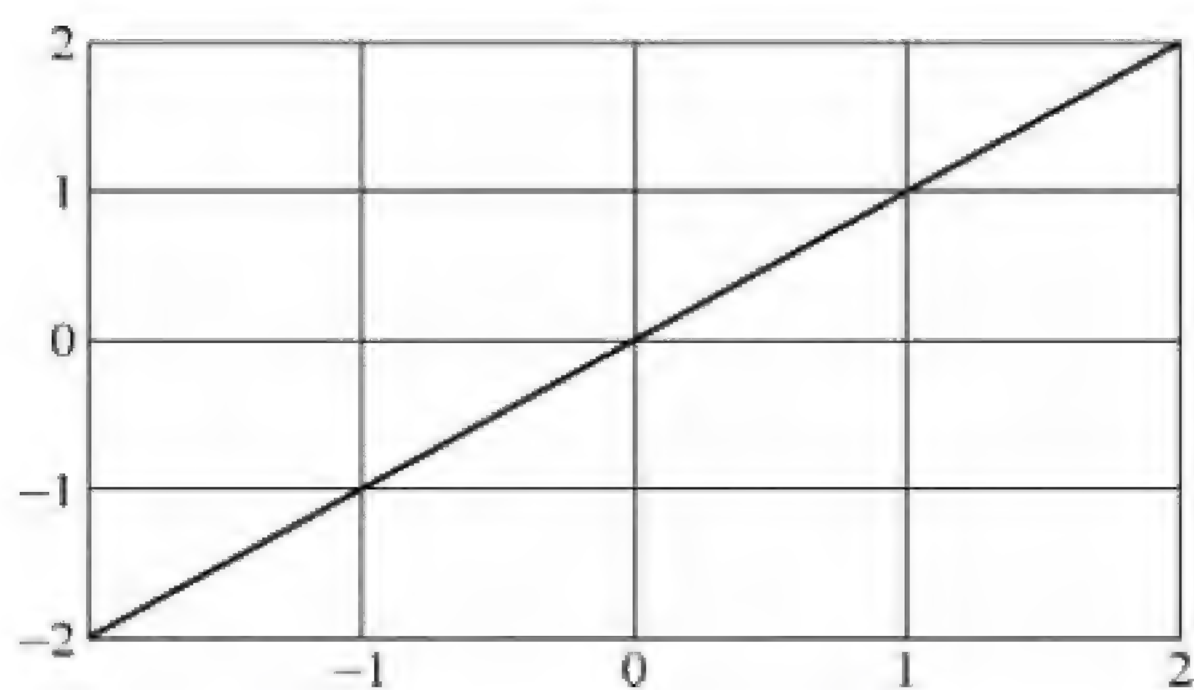


图 16-3 purelin 函数图形

其中，N 为输入向量，FP 为功能参数(可省略)。

函数的 MATLAB 应用示例如下：

```
clear all
clc
a = -2:0.05:2;
b = logsig(a);
plot(a,b)
grid on
```

运行后，得到如图 16-4 所示的 logsig 函数图形。

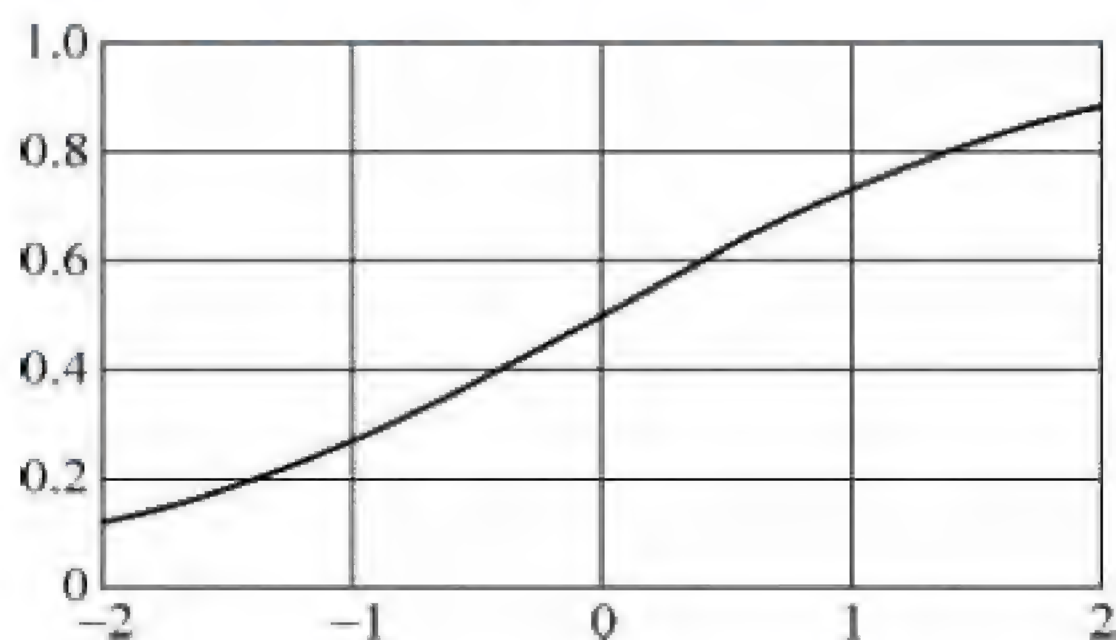


图 16-4 logsig 函数图形

5. tansig 函数

该函数是一个双正切函数。其调用格式如下：

```
A = tansig (N,FP)
```

其中，N 为输入向量，FP 为功能参数(可省略)。

函数的 MATLAB 应用示例如下：

```
clear all
clc
a = -2:0.05:2;
b = tansig(a);
```

```
plot(a,b)
grid on
```

运行后，得到如图 16-5 所示的 tansig 函数图形。

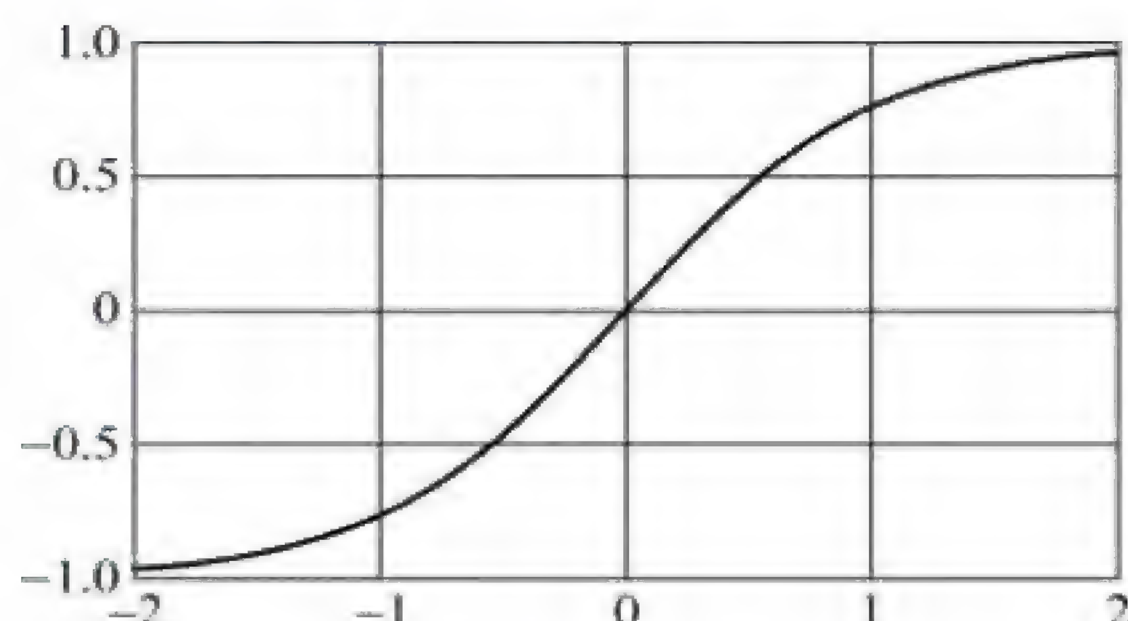


图 16-5 tansig 函数图形

【例 16-1】 一个具有 tansig 激活函数的单层网络，输入矢量有 4 组，每组有 4 个分量，输出矢量有 5 个神经元。假定输入矢量和权矢量均取值为 1，现使用 MATLAB 计算神经网络的输出。

解：在 M 文件编辑器中输入如下代码：

```
clear all
clc
a = 4;          % 列数
b = 4;          % 行数
c = 5;          % 神经元个数
W = ones(c,b);
B = ones(c,a);
P = ones(b,a);
x = W * P + B;  % 计算神经网络加权输入
A = tansig(x)   % 计算神经网络输出
```

运行以上程序，得到输出结果如下：

```
A =
    0.9999   0.9999   0.9999   0.9999
    0.9999   0.9999   0.9999   0.9999
    0.9999   0.9999   0.9999   0.9999
    0.9999   0.9999   0.9999   0.9999
    0.9999   0.9999   0.9999   0.9999
```

16.2.2 神经网络通用函数

在 MATLAB 神经网络工具箱中，有些函数几乎可以用于所有种类的神经网络，比如神经初始化函数 init、神经网络仿真函数 sim 等，称为神经网络通用函数。本节介绍几个经典的神经网络通用函数。

1. 初始化函数 init

初始化函数 init 是对一个已经存在的神经网络参数进行初始化，即修正该网络的权值和偏值等参数。

该函数的调用格式为

```
net = init(NET)
```

其中，NET 是没有初始化的神经网络，net 是经过初始化的神经网络。

2. 单层神经网络初始化函数 initlay()

initlay()是对层-层结构神经网络进行初始化，即修正该网络的权值和偏值。其调用格式如下：

```
net = initlay(NET)
```

其中，NET 是没有初始化的神经网络，net 是经过初始化的神经网络。

3. 权值和偏值初始化函数 initwb()

initwb()函数可以对神经网络的某一层权值、偏值进行初始化修正，该神经网络的每层权值和偏值按照预先设定的修正方式来完成。其调用格式为

```
net = initwb(NET, i)
```

其中，NET 是没有初始化的神经网络，i 为需要进行权值和偏值进行修正的，net 是第 i 层经过初始化的神经网络。

4. 训练函数 train()

train()是一种通用的学习函数，可以训练一个神经网络。训练函数作用就是不断重复地把一组输入向量输入到某一个神经网络中，实时更新神经网络的权值、偏值，当神经网络训练达到设定的最大学习步数、最小误差梯度或误差目标等条件后，停止训练。

函数调用格式如下：

```
[net, tr, Y, E] = train(NET, X, T, Xi, Ai)
```

其中，NET 为需要训练的神经网络；

X 为神经网络的输入；

T 为训练神经网络的目标输出，默认值为 0；

Xi 表示初始输入延时，默认值为 0；

Ai 表示初始的层延时，默认值为 0；

net 表示完成训练的神经网络；

tr 表示神经网络训练的步数
Y 为神经网络的输出；
E 表示网络训练误差。

5. 仿真函数 sim

神经网络完成训练后，其权值和偏值也确认了。利用 sim 函数可以检测已经完成训练的神经网络的性能。其调用格式为

```
[Y,Xf,Af,E] = sim(net,X,Xi,Ai,T)
```

其中，net 是要训练的网络；
X 为神经网络的输入；
Xi 表示初始输入延时，默认值为 0；
Ai 表示初始的层延时，默认值为 0；
T 为训练神经网络的目标输出，默认值为 0；
net 表示完成训练的神经网络；
tr 表示神经网络训练的步数
Y 为神经网络的输出；
Xf 为最终输入延时；
Af 为最终的层延时；
E 表示网络误差。

6. 和函数 netsum

神经网络输入的和函数是通过某一层的加权输入和偏值相加作为该层的输入。调用格式如下：

```
N = netsum({Z1,Z2,…,Zn},FP)
```

其中，Z_n 是 S×Q 维矩阵。

7. 权值点积函数 dotprod

神经网络输入向量与权值的点积可以得到加权输入。该函数调用格式如下：

```
Z = dotprod(W,P,FP)
```

其中，W 为权值矩阵，P 为输入向量，FP 为功能参数(可省略)，FP 为 Z 为权值矩阵与输入向量的点积。

8. 网络输入的积函数 netprod

该函数是将神经网络某一层加权输入和偏值相乘的结果，作为该层的输入。其调用格式为

```
N = netprod({Z1,Z2, …,Zn})
```

其中，Z_n 为 Z×Q 维矩阵。

16.2.3 神经网络的 MATLAB 实现

神经网络是由基本神经元相互连接形成的，能模拟人的神经处理信息的方式，也可以解决很多利用传统方法无法解决的难题。利用 MATLAB 工具箱函数可以完成各种神经网络的设计、训练和仿真。

本节重点介绍几种经典的 MATLAB 神经网络应用。

1. BP 神经网络在函数逼近中的应用

BP 神经网络算法是在 BP 神经网络现有算法的基础上提出的，是通过任意选定一组权值，将给定的目标输出直接作为线性方程的代数和来建立线性方程组求解，不存在传统方法的局部极小及收敛速度慢的问题，且更易理解。

BP 网络有很强的映射能力，主要用于模式识别分类、函数逼近、函数压缩等。下面将通过实例来说明 BP 网络在函数逼近方面的应用。

【例 16-2】 使用 MATLAB 设计一个 BP 神经网络，要求逼近以下函数：

$$g(x) = 1 + \sin(k\pi/2x)$$

其中，分别令 $k=1$、5、9 进行仿真，通过调节参数得出信号的频率与隐含层节点之间、隐含层节点与函数逼近能力之间的关系。

解：假设频率参数 $k=1$，绘制要逼近的非线性函数的目标曲线。MATLAB 代码如下：

```
clear all
clc
k=1;
p=[-2:0.05:7];
t=1+cos(k*pi/2*p);
plot(p,t,'-');
title('逼近非线性函数');
xlabel('时间');
ylabel('非线性函数');
```

运行代码后，得到目标曲线如图 16-6 所示。

用 MATLAB 神经网络工具箱的 newff 函数建立 BP 网络结构。其中，隐含层神经元数目 n 暂设为 7，输出层有一个神经元。选择隐含层和输出层神经元传递函数分别为 tansig 函数和 purelin 函数，网络训练的算法采用 Levenberg-Marquardt 算法 trainlm。

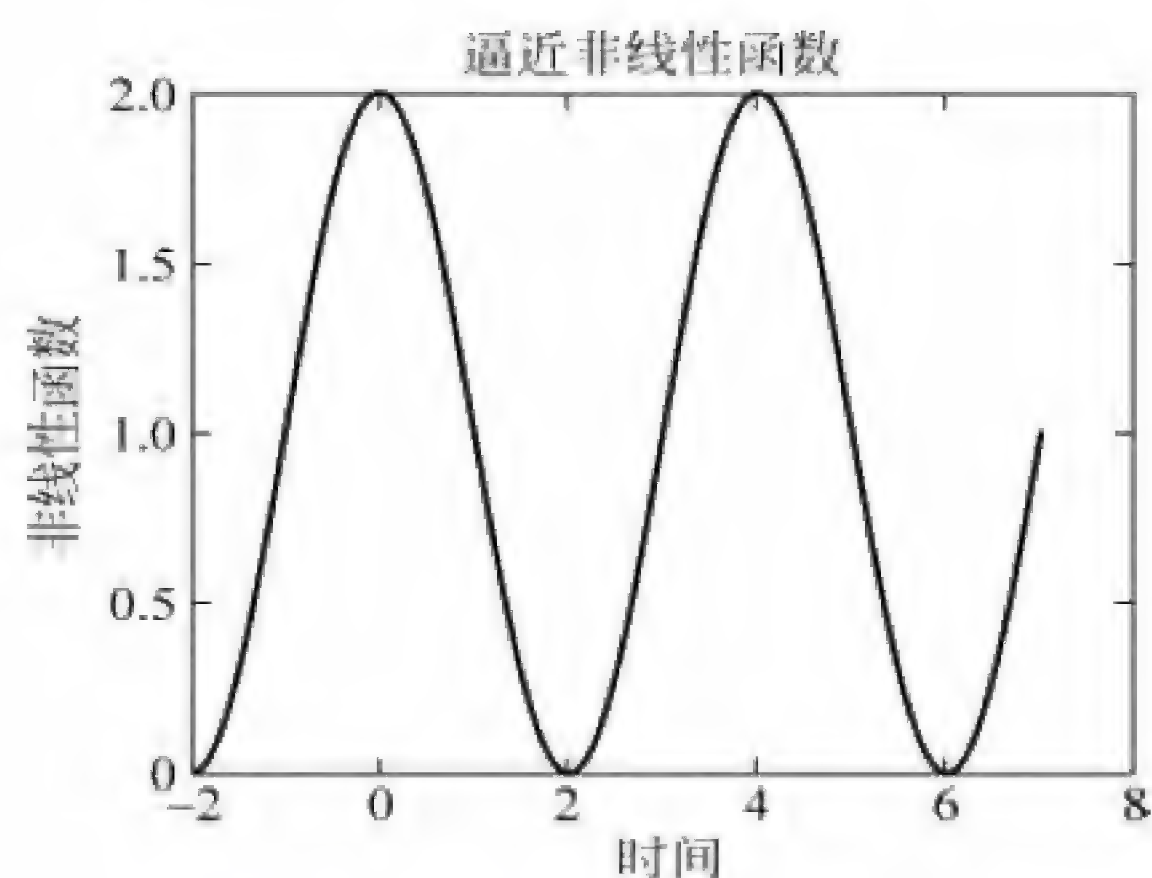


图 16-6　逼近的非线性函数曲线

```
n = 7;
net = newff(minmax(p),[n,1],{'tansig' 'purelin'},'trainlm');
y1 = sim(net,p);
figure;
plot(p,t,'-',p,y1,':')
title('未训练网络的输出结果');
xlabel('时间');
ylabel('仿真输出——原函数-');
```

运行后得到网络输出曲线与原函数的比较图,如图 16-7 所示。

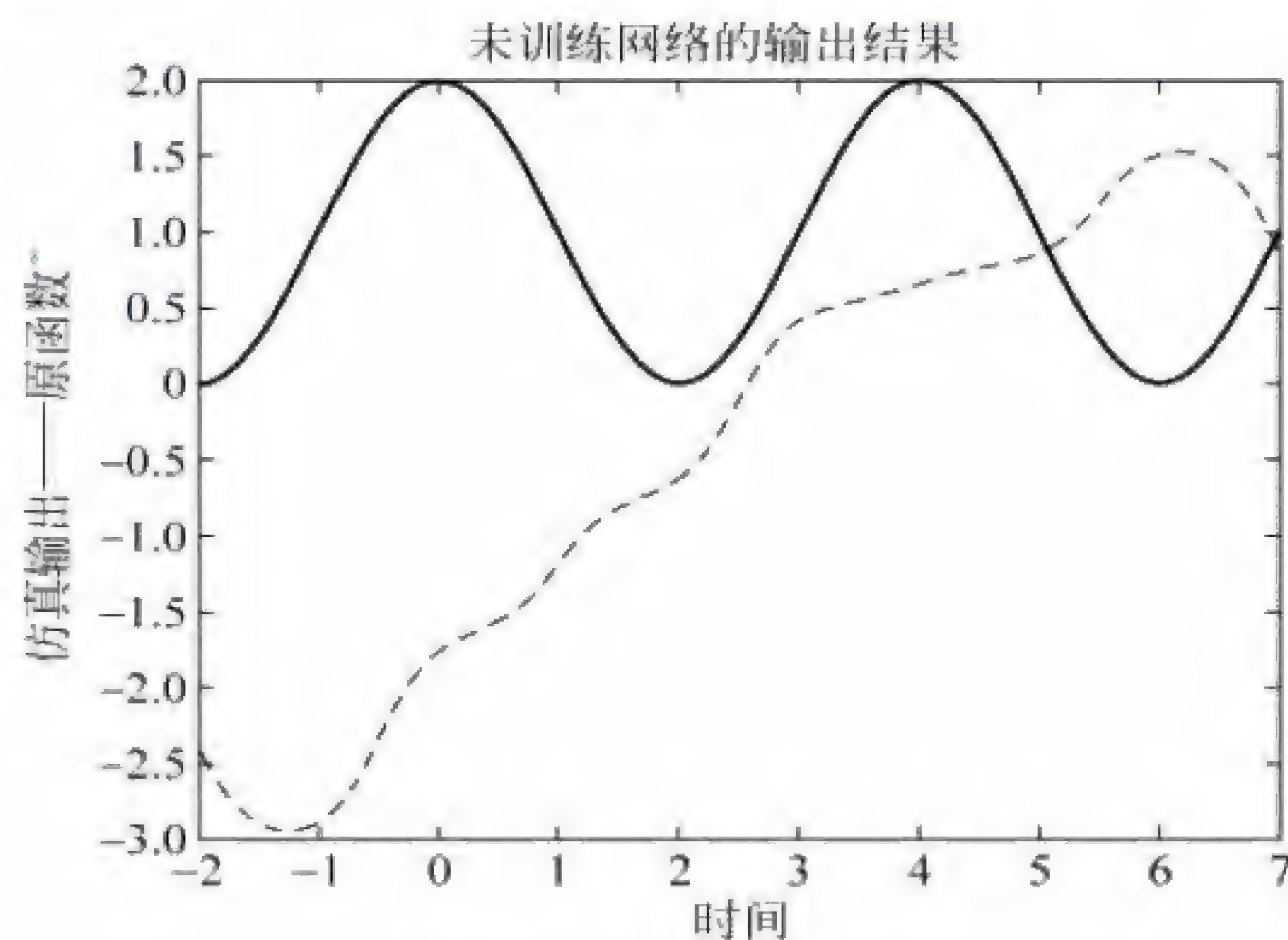


图 16-7　网络输出曲线与原函数的比较图

因为使用 newff()函数建立函数网络时,权值和阈值的初始化是随机的,所以网络输出结构很差,根本达不到函数逼近的目的,每次运行的结果也不同。

在 MATLAB 中使用 train()函数对神经网络进行训练之前,需要预先设置网络训练参数。将训练时间设置为 200,训练精度设置为 0.2,其余参数使用默认值。训练神经网络的 MATLAB 代码如下:

```
net.trainParam.epochs = 300;          % 网络训练时间设置为 300
net.trainParam.goal = 0.3;            % 网络训练精度设置为 0.3
net = train(net,p,t);                 % 开始训练网络
```

运行后得到训练结果如图 16-8 所示。从图中可以看出，神经网络运行一步网络输出误差就达到设定的训练精度。

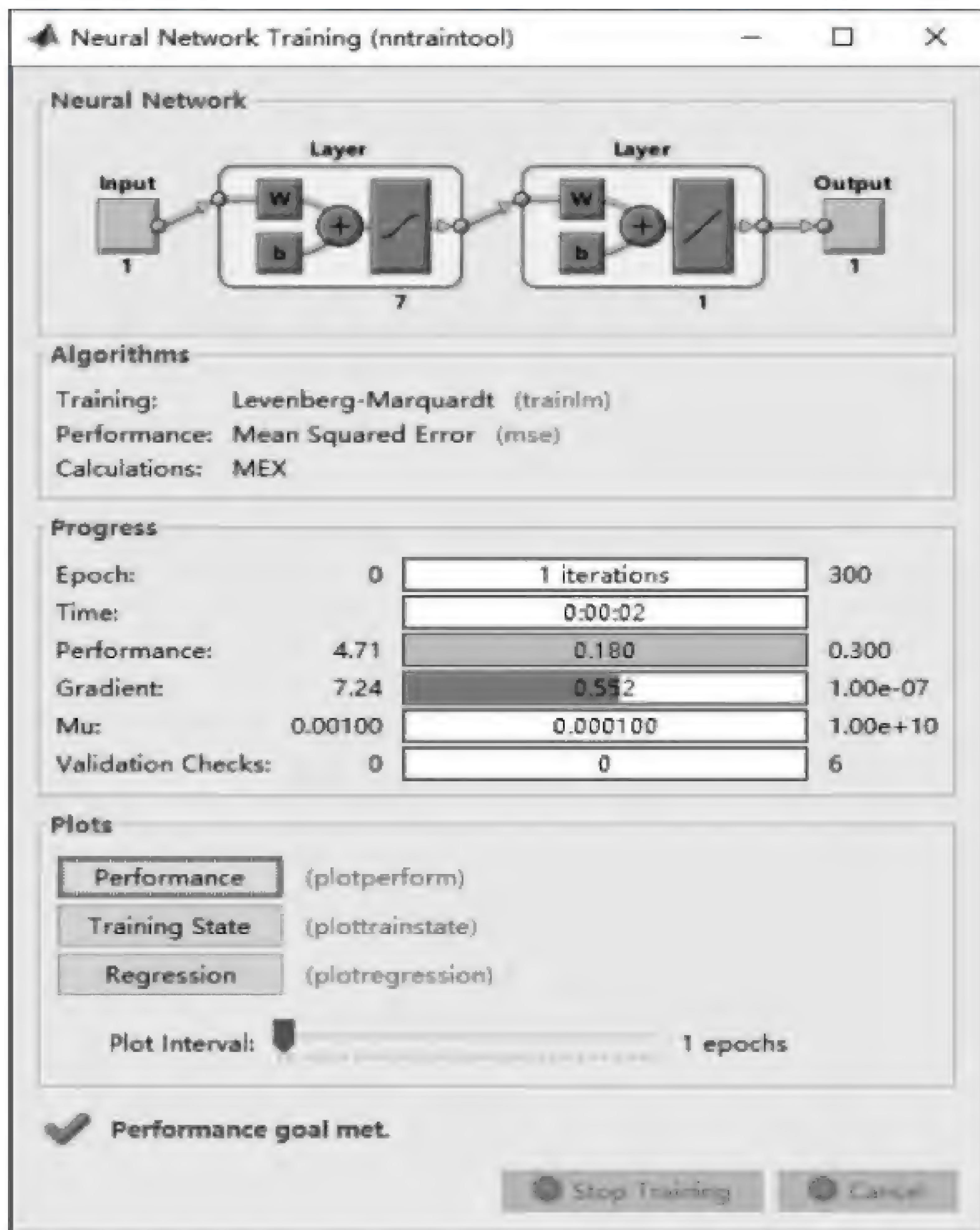


图 16-8　训练结果

对于训练好的网络进行仿真

```
y2 = sim(net,p);
figure(1);
plot(p,t,'-',p,y1,':',p,y2, '--')
title('训练前后结果');
xlabel('时间');
ylabel('仿真输出');
```

绘制网络输出曲线，并与原始非线性函数曲线以及未训练网络的输出结果曲线相比较，结果如图 16-9 所示。

相对于没有训练的曲线，经过训练之后的曲线和原始的目标曲线更接近。这说明经

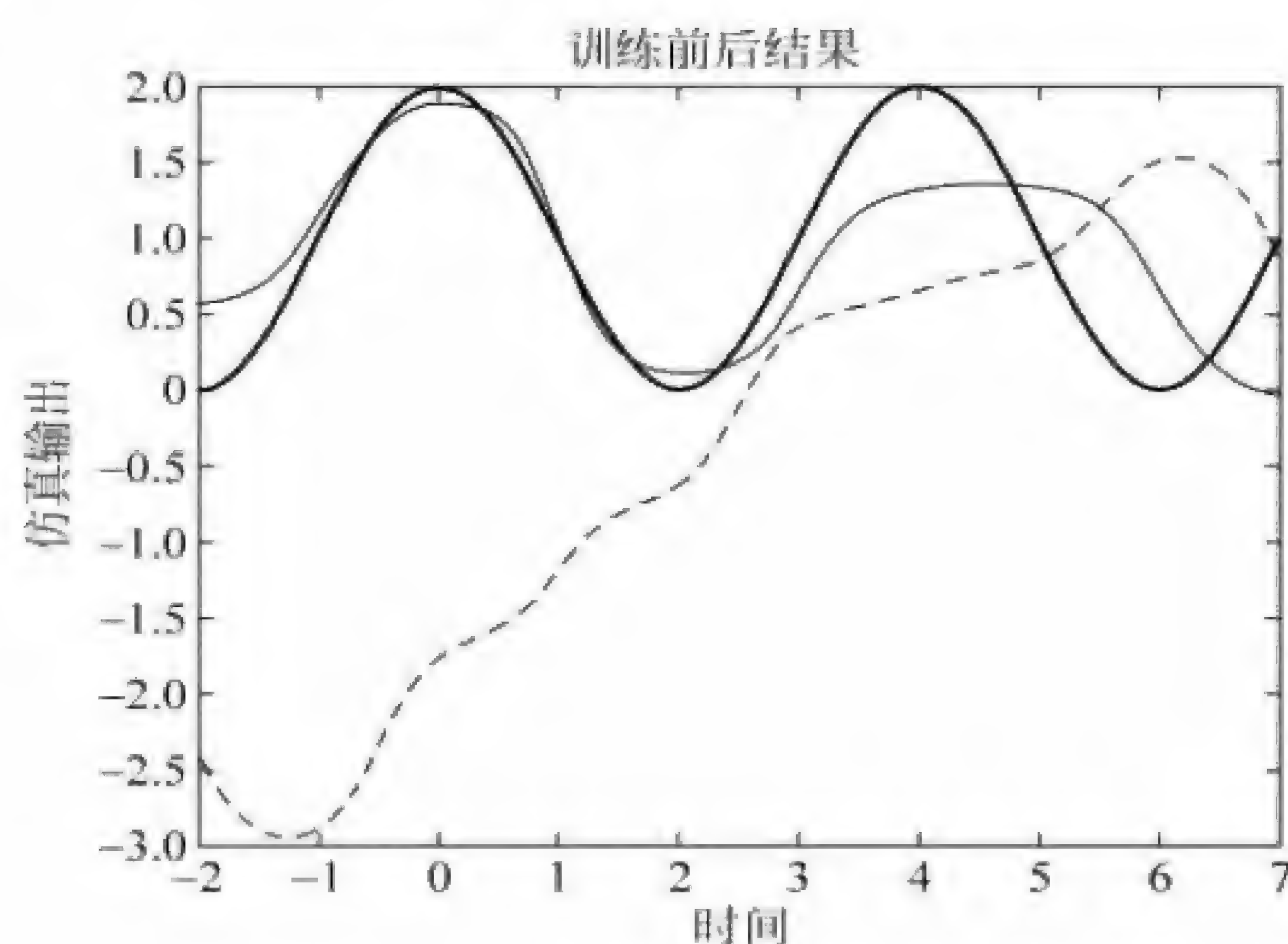


图 16-9 训练后网络的输出结果

过训练后，BP 网络对非线性函数的逼近效果比较好。

改变非线性函数的频率和 BP 函数隐含层神经元的数目，对于函数逼近的效果有一定的影响。

当频率参数设为 $k=5$，隐含层神经元数目保持不变，编写以下 MATLAB 代码：

```
clear all
clc
k = 5;
p = [ - 2:0.05:7];
t = 1 + cos(k * pi/2 * p);
figure(1);
plot(p,t,' - ');
title('逼近非线性函数');
xlabel('时间');
ylabel('非线性函数');

n = 7;
net  =  newff(minmax(p),[n,1],{'tansig' 'purelin'},'trainlm');
y1 = sim(net,p);
figure(2);
plot(p,t,' - ',p,y1,':')
title('未训练网络的输出结果');
xlabel('时间');
ylabel('仿真输出——原函数');

net.trainParam.epochs = 300;              % 网络训练时间设置为 300
net.trainParam.goal = 0.3;                % 网络训练精度设置为 0.3
net = train(net,p,t);                     % 开始训练网络

y2 = sim(net,p);
figure(3);
```

```
plot(p,t,'-',p,y1,':',p,y2, '--')
title('训练前后结果');
xlabel('时间');
ylabel('仿真输出');
```

运行代码后，得到训练前后结果如图 16-10 所示。

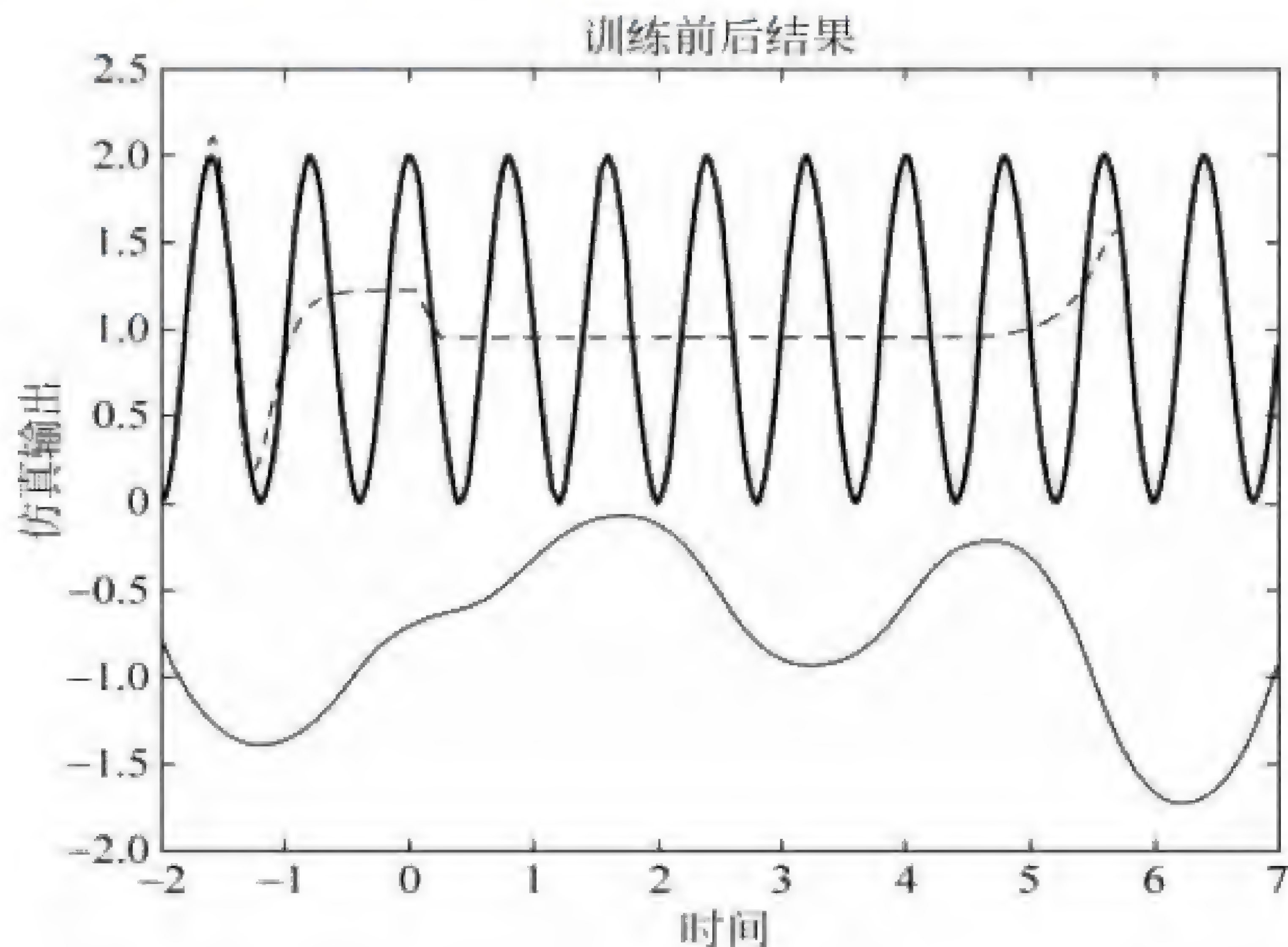


图 16-10　当 $k=5$ 时训练后网络的输出结果

当频率参数设为 $k=9$，隐含层神经元数目保持不变，编写以下 MATLAB 代码：

```
clear all
clc
k = 9;
p = [ - 2:0.05:7];
t = 1 + cos(k * pi/2 * p);
figure(1);
plot(p,t,'-');
title('逼近非线性函数');
xlabel('时间');
ylabel('非线性函数');

n = 7;
net = newff(minmax(p),[n,1],{'tansig' 'purelin'},'trainlm');
y1 = sim(net,p);
figure(2);
plot(p,t,'-',p,y1,':')
title('未训练网络的输出结果');
xlabel('时间');
ylabel('仿真输出——原函数');

net.trainParam.epochs = 300;          % 网络训练时间设置为 300
net.trainParam.goal = 0.3;            % 网络训练精度设置为 0.3
```

```
net = train(net,p,t);          % 开始训练网络

y2 = sim(net,p);
figure(3);
plot(p,t,'-',p,y1,':',p,y2, '--')
title('训练前后结果');
xlabel('时间');
ylabel('仿真输出');
```

运行代码后，得到结果如图 16-11 所示。

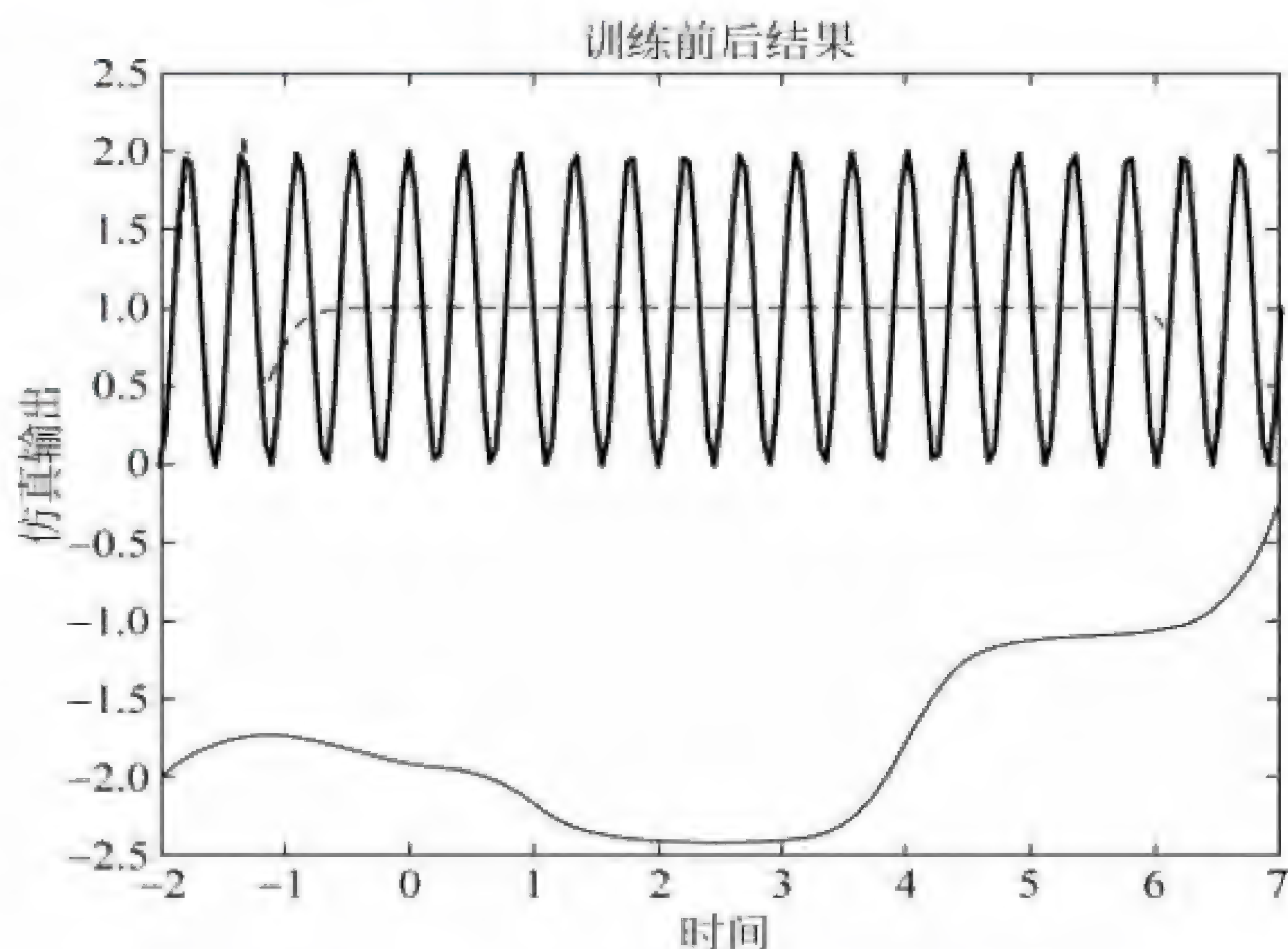


图 16-11　当 $k=9$ 时训练后网络的输出结果

通过上述仿真结果可知，当 $k=9$ 时，BP 神经网络对函数取得了较好的逼近效果，k 取不同的值对函数逼近的效果有很大的影响。

2. RBF 神经网络在函数曲线拟合中的应用

曲线拟合是用连续曲线近似地刻画或比拟平面上离散点组所表示的坐标之间的函数关系的一种数据处理方法，是用解析表达式表示离散数据的一种方法。

RBF 神经网络具有良好的推广能力，在完成函数拟合任务时速度最快，结构也比较简单，对具有复杂函数关系的问题作泛函逼近时，具有较高的精确度。

【例 16-3】 利用 MATLAB 神经网络工具箱，建立 RBF 网络拟合未知函数 $y=18+x_1^2-8\sin(2\pi x_1)+5x_2^2-3\cos(2\pi x_2)$。

解： 按照 RBF 神经网络的结原理，首先需要随机产生输入变量 x_1、x_2，并根据产生的输入变量和未知函数求得输出变量 y。

将输入变量 x_1、x_2 和输出变量 y 作为 RBF 网络的输入数据和输出数据，建立近似和精确 RBF 网络进行回归分析，并评价拟合效果。

建立 RBF 神经网络的 MATLAB 代码如下：

```
clc
clear all

x1 = -2:0.02:2;
x2 = -2:0.02:2;

y = 18 + x1.^2 - 8 * sin(2 * pi * x1) + 5 * x2.^2 - 3 * cos(2 * pi * x2);

net = newrbe([x1;x2],y)
t = sim(net,[x1;x2]);
figure(1)
plot3(x1,x2,y,'rd');
hold on;
plot3(x1,x2,t,'b-.');
view(100,25)
title('拟合效果')
xlabel('x1')
ylabel('x2')
zlabel('y')
grid on
```

运行代码，在 MATLAB 的命令窗口会显示程序建立 RBF 网络的相关信息，具体如下：

```
> In newrbe>designrbe (line 138)
  In newrbe>create_network (line 122)
  In newrbe (line 64)
  In ex16_3 (line 9)
警告: 秩不足，秩 = 24,tol = 6.359012e-13.

net =
    Neural Network
              name: 'Radial Basis Network, Exact'
          userdata: (your custom info)

    dimensions:
         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 805
        sampleTime: 1

    connections:
       biasConnect: [1; 1]
      inputConnect: [1; 0]
```

```
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:
             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:
          adaptFcn: (none)
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: (none)
       divideParam: (none)
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {}
        plotParams: {1x0 cell array of 0 params}
          trainFcn: (none)
        trainParam: (none)

    weight and bias values:
                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:
             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs
```

运行代码后，得到拟合效果如图 16-12 所示。

继续利用 approximateRBF 神经网络对例 16-3 中的函数进行拟合，具体 MATLAB 代码如下：

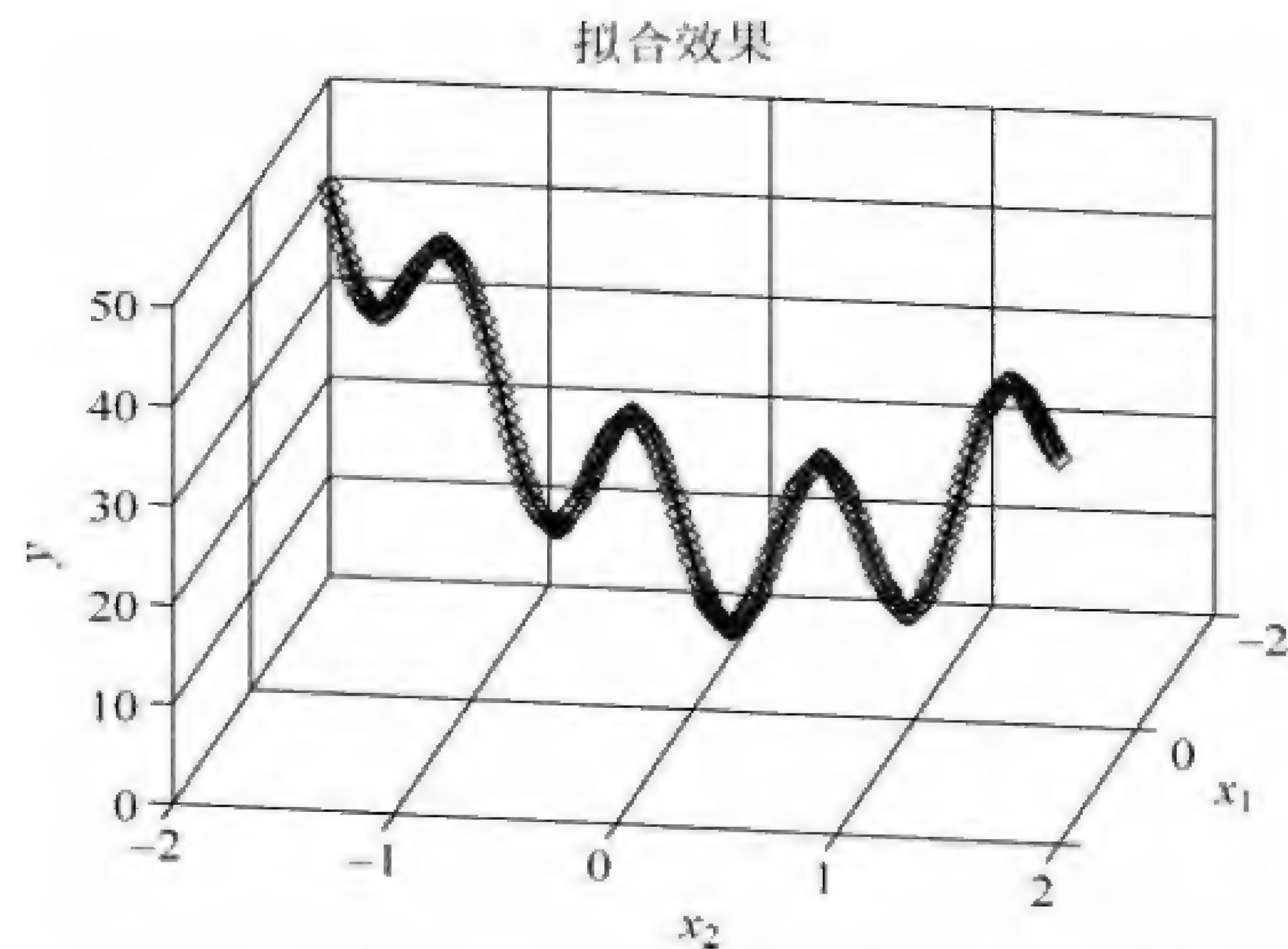


图 16-12　拟合效果图

```
clc
clear all

x = rand(2,300);
x = (x - 0.5) * 1 * 2;
x1 = x(1,:);
x2 = x(2,:);

y = 18 + x1.^2 - 8 * sin(2 * pi * x1) + 5 * x2.^2 - 3 * cos(2 * pi * x2);

net = newrb(x,y);
[n,m] = meshgrid( - 1:0.1:1);
row = size(n);
tx1 = n(:);
tx1 = tx1';
tx2 = m(:);
tx2 = tx2';
tx = [tx1;tx2];

t = sim(net,tx);
p = reshape(t,row);
subplot(1,3,1)
mesh(n,m,p);
zlim([0,50])
title('仿真结果图像')
%目标函数图像
[x1, x2] = meshgrid( - 1:0.1:1);
y = 30 + x1.^2 - 5 * cos(2 * pi * x1) + 3 * x2.^2 - 5 * cos(2 * pi * x2);
subplot(1,3,2)
mesh(x1,x2,y);
zlim([0,50])
title('目标函数图像')
%目标函数图像和仿真函数图像的误差图像
```

```
subplot(1,3,3)
mesh(x1,x2,y - p);
zlim([ - 0.1,0.05])
title('误差图像')
set(gcf,'position')
```

运行代码后,得到仿真结果如图 16-13 所示。

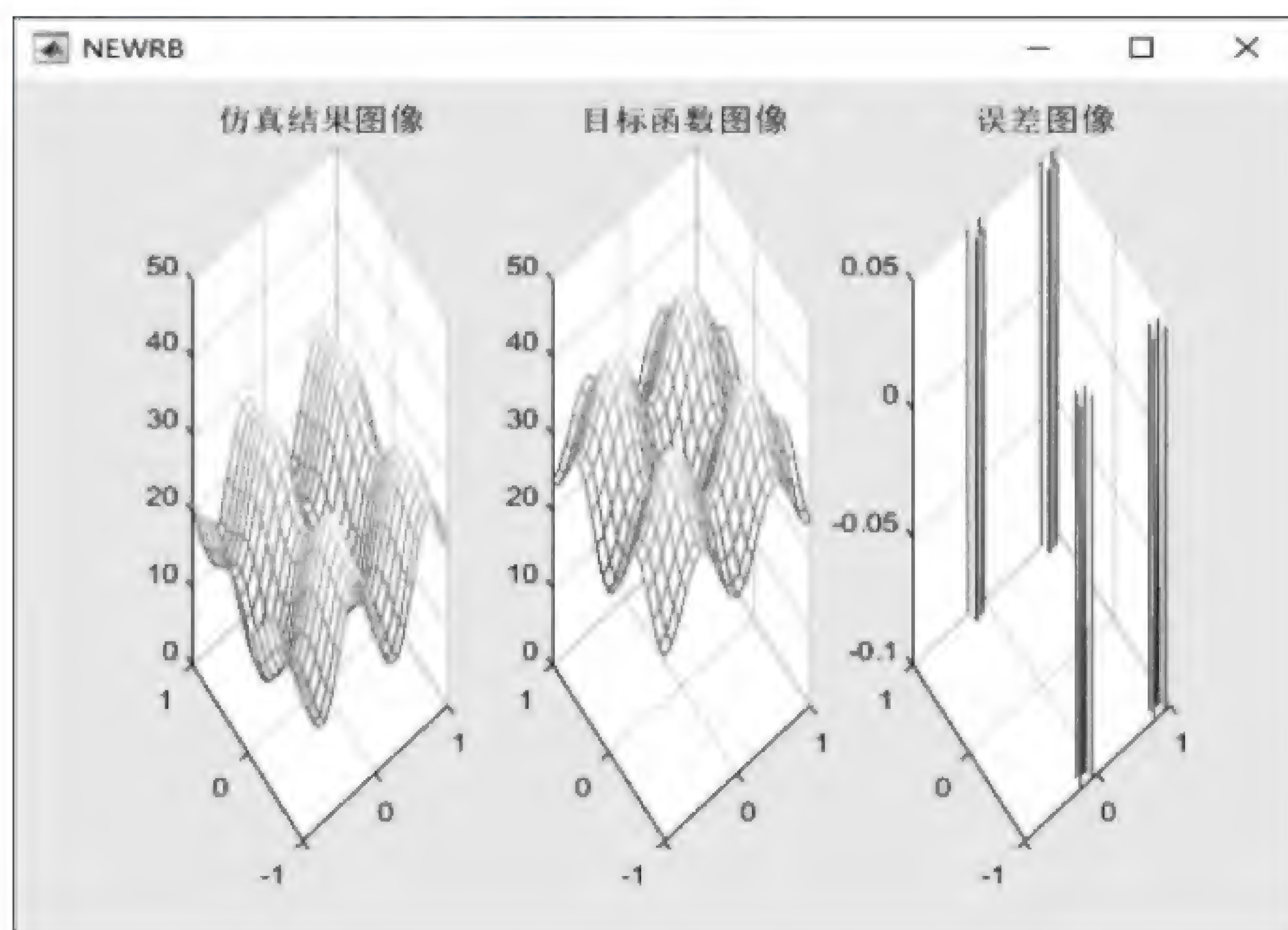


图 16-13　用 approximateRBF 神经网络对例 16-3 中的函数进行拟合的结果

在 MATLAB 命令窗口显示的 RBF 网络仿真输出和目标函数值之间的差值,具体如下:

```
NEWRB, neurons = 0, MSE = 38.3208
NEWRB, neurons = 50, MSE = 0.0139403
NEWRB, neurons = 100, MSE = 1.2775e - 06
NEWRB, neurons = 150, MSE = 3.39522e - 07
NEWRB, neurons = 200, MSE = 1.17366e - 07
NEWRB, neurons = 250, MSE = 5.88275e - 08
NEWRB, neurons = 300, MSE = 4.21626e - 08
  空的 0×0 cell 数组
```

3. Hopfield 神经网络在稳定平衡点中的应用

Hebb 学习规则是 Hopfield 神经网络常用的学习算法,该算法多用于在控制系统设计中求解约束优化问题,此外在系统的辨识中也有广泛的应用。

【例 16-4】　假设有 2 个目标向量,使用 newhop 函数建立具有 2 个稳定点的 Hopfield 神经网络。

解:根据题中所假设的条件,建立 Hopfield 神经网络 MATLAB 代码如下:

```
clear all
clc
% 定义具有 2 列的目标向量
T = [-1 1; 1 -1; -1 -1];

% 绘制 Hopfield 神经网络稳定空间图形
axis([-1 1 -1 1 -1 1])
set(gca,'box','on');
axis manual;
hold on;
plot3(T(1,:),T(2,:),T(3,:),'r*')
title('Hopfield Network State Space')
xlabel('x(1)');
ylabel('x(2)');
zlabel('x(3)');
view([35 16 50]);
```

两个稳定点的 Hopfield 神经网络稳定空间图形如图 16-14 所示。

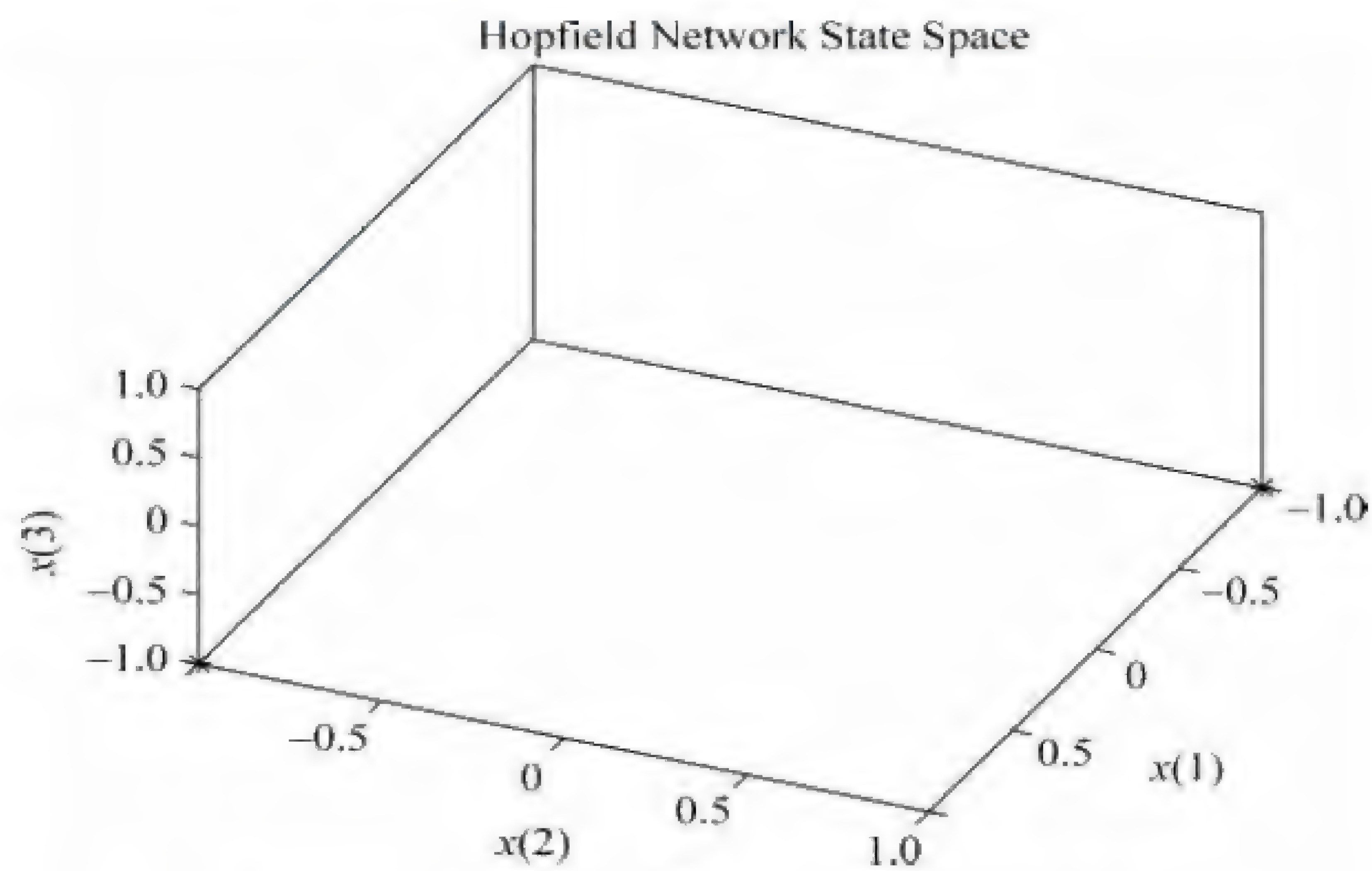


图 16-14　Hopfield 神经网络稳定空间图形

```
% 创建 Hopfield 神经网络
net = newhop(T);

% 随机起始点
a = {rands(3,1)};
% 设定 Hopfield 仿真参数
[y,Pf,Af] = net({1 10},{},a);

% 设定一个稳定空间内的活动点
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
hold on
plot3(start(1,1),start(2,1),start(3,1),'bx', ...
    record(1,:),record(2,:),record(3,:))
```

在稳定空间内设定一个活动点的图形如图 16-15 所示。

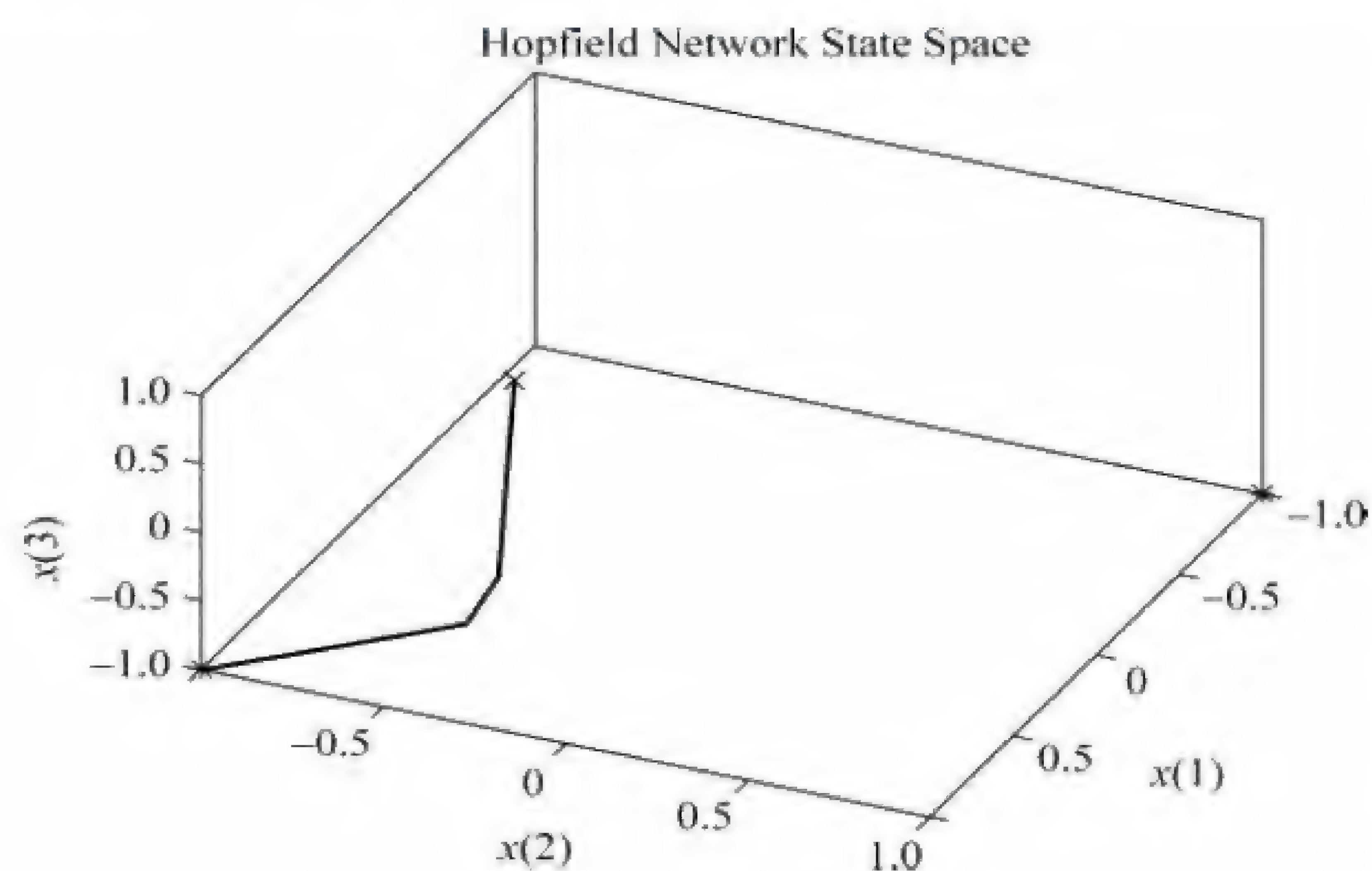


图 16-15　在稳定空间内设定一个活动点

```
% 循环模拟 30 个初始条件
color = 'rgbmy';
for i = 1:30
   a = {rands(3,1)};
   [y,Pf,Af] = net({1 10},{},a);
   record = [cell2mat(a) cell2mat(y)];
   start = cell2mat(a);
   plot3(start(1,1),start(2,1),start(3,1),'kx', ...
      record(1,:),record(2,:),record(3,:),color(rem(i,5) + 1))
end
```

循环模拟 30 个起始点得到的稳定空间如图 16-16 所示。

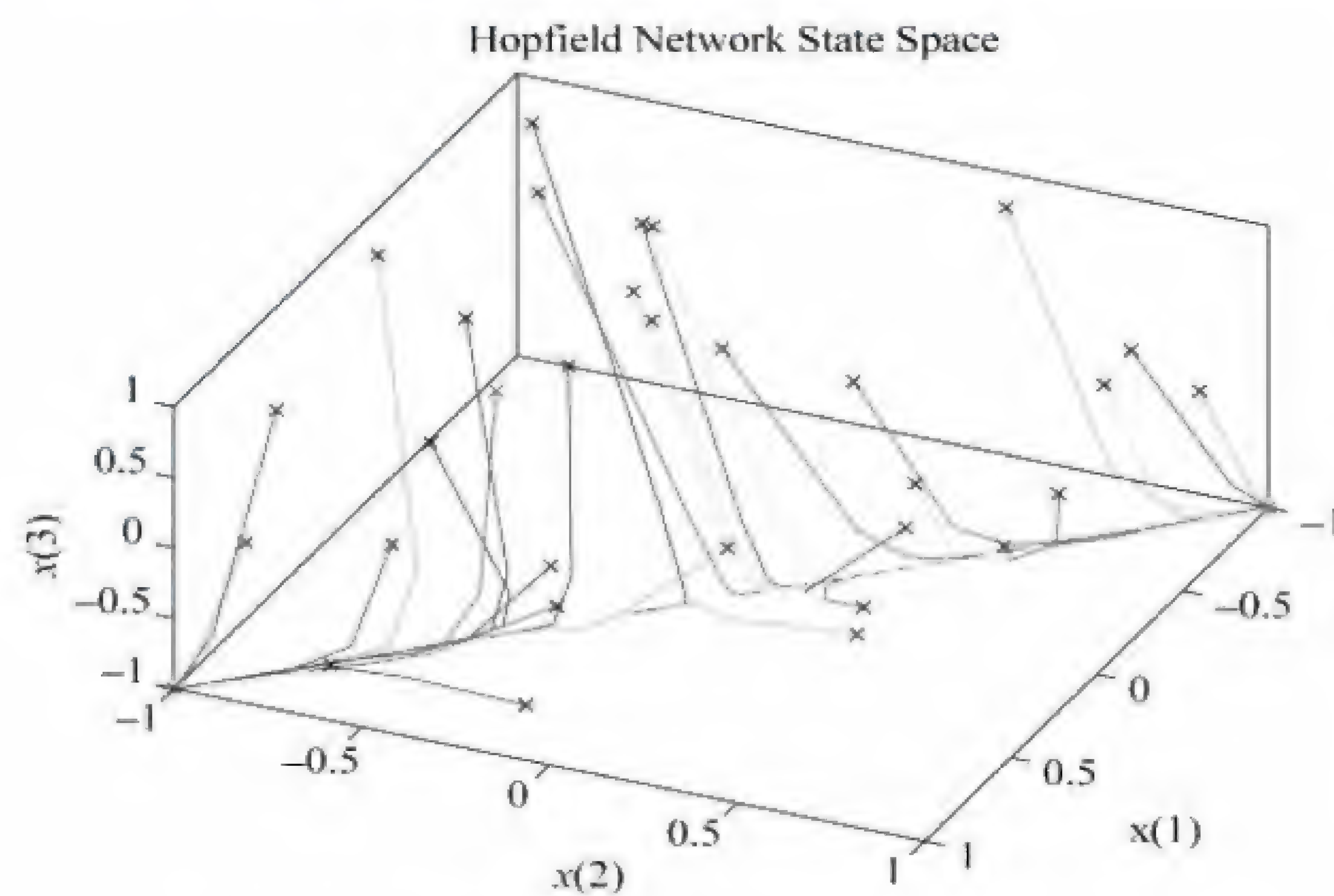


图 16-16　重复模拟 30 个起始点得到的稳定空间

```
%使用输入向量P仿真Hopfield神经网络
P = [ 0.1   -1   -0.6    1    1    1; ...
        0    0     0    0    0    0; ...
     -0.1    1    0.6   -1   -1   -1];
cla
plot3(T(1,:),T(2,:),T(3,:),'r*')
color = 'rgbmy';
for i = 1:6
   a = {P(:,i)};
   [y,Pf,Af] = net({1 10},{},a);
   record = [cell2mat(a) cell2mat(y)];
   start = cell2mat(a);
   plot3(start(1,1),start(2,1),start(3,1),'kx', ...
      record(1,:),record(2,:),record(3,:),color(rem(i,5)+1))
end
```

运行代码后，得到两个目标稳定点之间的起始点都进入稳定空间的中心如图 16-17 所示。

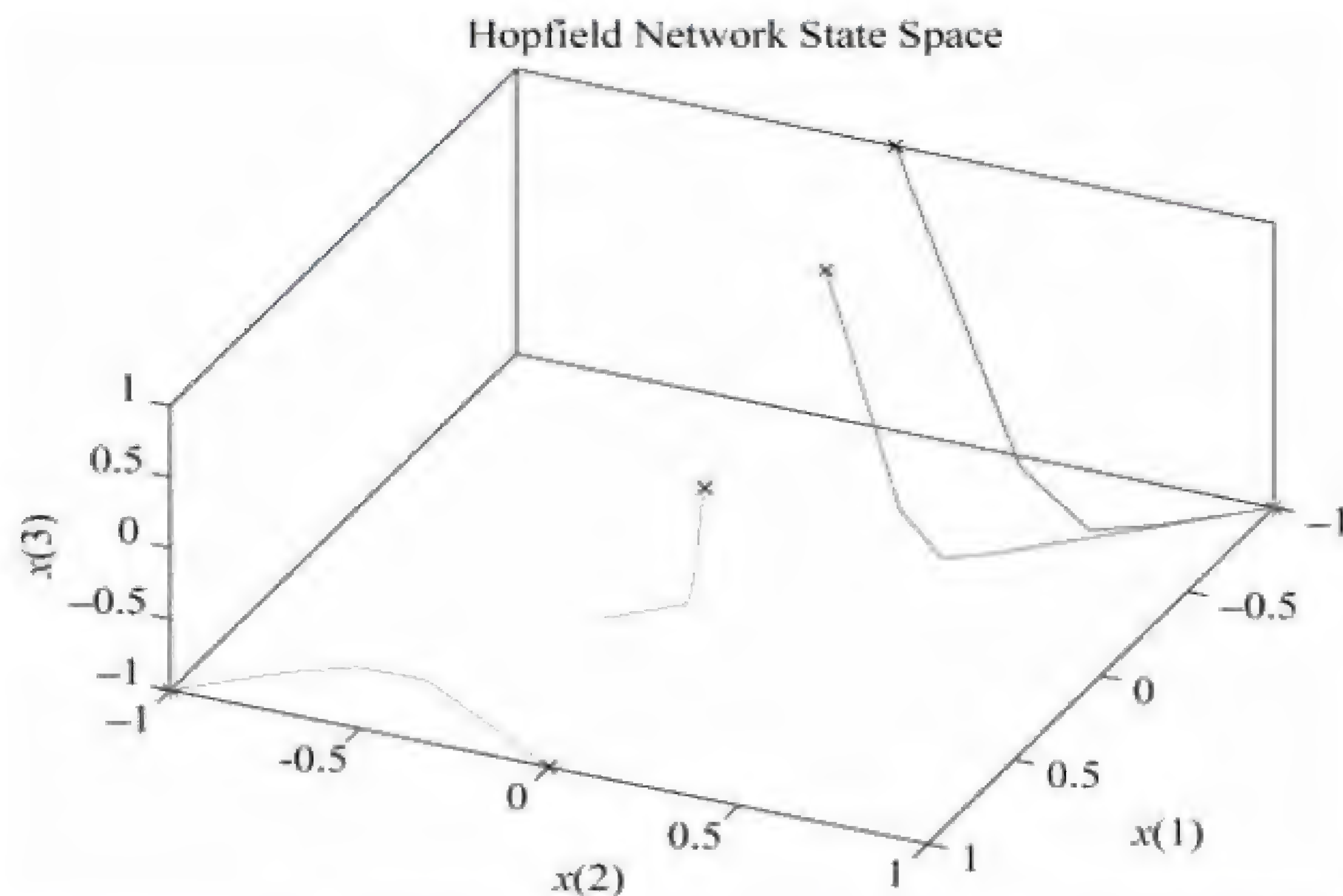


图 16-17　两个目标稳定点之间的起始点都进入稳定空间中心

4. 自组织特征映射神经网络在数据分类中的应用

生物学研究表明，在人脑的感觉通道上，神经元的组织是有序排列的。大脑皮层中神经元的响应特点不是先天安排好的，而是通过后天的学习自组织形成的。

自组织神经网络是无导师学习网络。它通过自动寻找样本中的内在规律和本质属性，自组织、自适应地改变网络参数与结构。

自组织特征映射神经网络是一个神经网络接受外界输入模式时，分为不同的对应区域，各区域对输入模式有不同的响应特征，而这个过程是自动完成的。其通常作为一种样本特征检测器，在样本排序、样本分类及样本检测方面有广泛的应用。

【例 16-5】 随机设定 13 个数据样本，每个样本包含 7 个数据，设计一个 SOM 网络对不同特性的数据进行分类。

解：根据题意设置，编写 MATLAB 代码如下：

```
clear all
clc
%设样本数据
m = 13;
n = 7;
P = rand(m,n);
for i = 1:m
    for j = 1:n
        if P(i,j)> 0.7
            P(i,j) = 1;
        else
            P(i,j) = 0;
        end
    end
end
P = P';

%newsom 建立 SOM 网络,竞争层为 7*9=63 个神经元
net = newsom(minmax(P),[7 9]);
plotsom(net.layers{1}.positions)

%4 种不同训练步数的仿真
a = [5 20 40 80 160 320 640 1280];
%随机初始化一个向量
yc = rands(13,13);

%训练步数为 5 时仿真
net.trainparam.epochs = a(1);
%训练网络和查看分类
net = train(net,P);
%仿真网络
y = sim(net,P);
yc(1,:) = vec2ind(y);
figure(1)
subplot(2,2,1);
plotsom(net.IW{1,1},net.layers{1}.distances)

%训练步数为 20 时仿真
net.trainparam.epochs = a(2);
%训练网络和查看分类
net = train(net,P);
%仿真网络
y = sim(net,P);
yc(2,:) = vec2ind(y);
subplot(2,2,2);
plotsom(net.IW{1,1},net.layers{1}.distances)
```

```
%训练步数为40时仿真
net.trainparam.epochs = a(3);
%训练网络
net = train(net,P);
%仿真网络
y = sim(net,P);
yc(3,:) = vec2ind(y);
subplot(2,2,3);
plotsom(net.IW{1,1},net.layers{1}.distances)

%训练步数为80时仿真
net.trainparam.epochs = a(4);
%训练网络
net = train(net,P);
%仿真网络
y = sim(net,P);
yc(4,:) = vec2ind(y);
subplot(2,2,4);
plotsom(net.IW{1,1},net.layers{1}.distances)
```

运行代码后得到神经网络训练图形如图16-18所示。

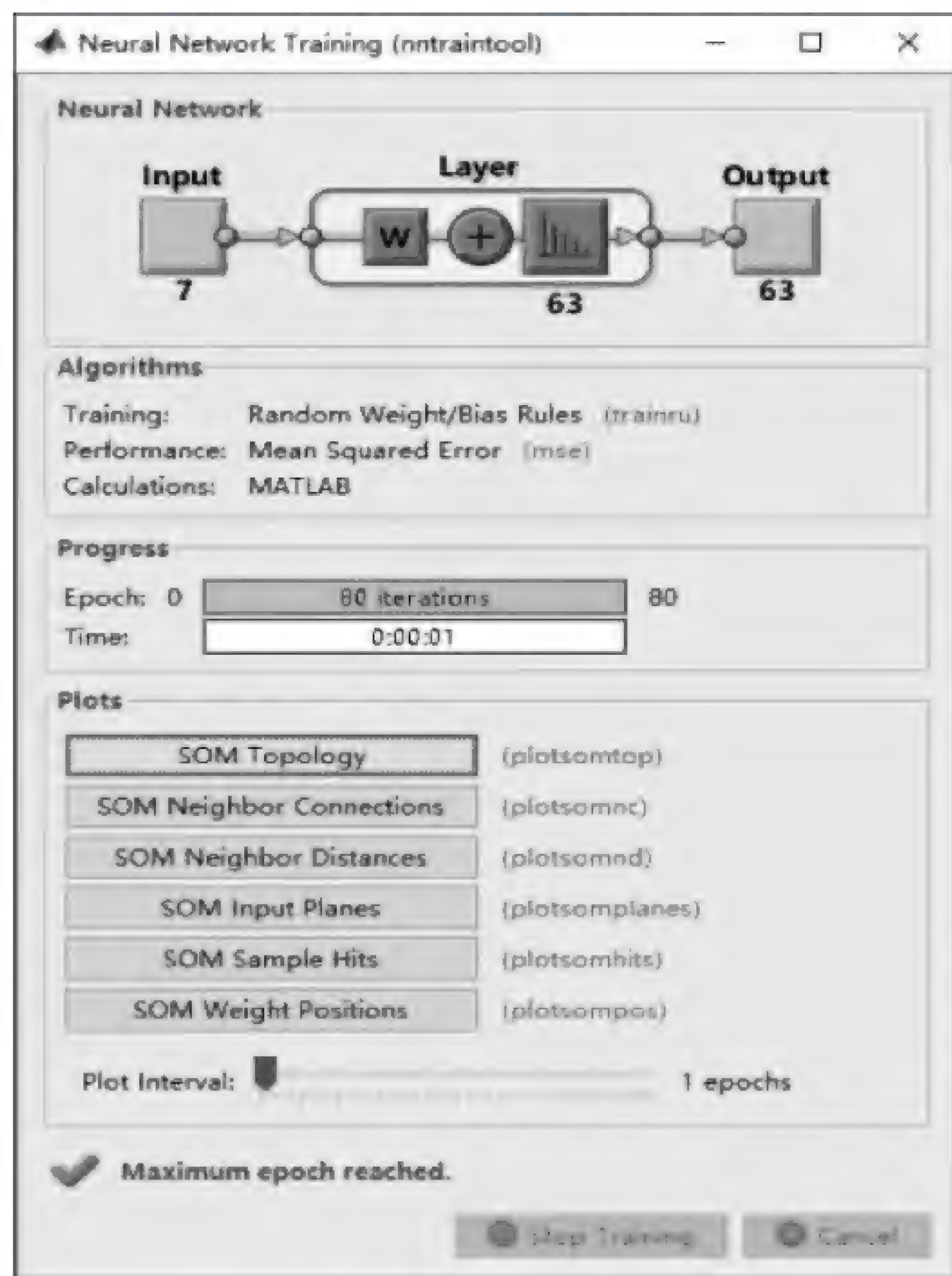


图16-18 神经元的位置示意图

训练 5、20、40 和 80 步神经网络，得到权值变化如图 16-19 所示。

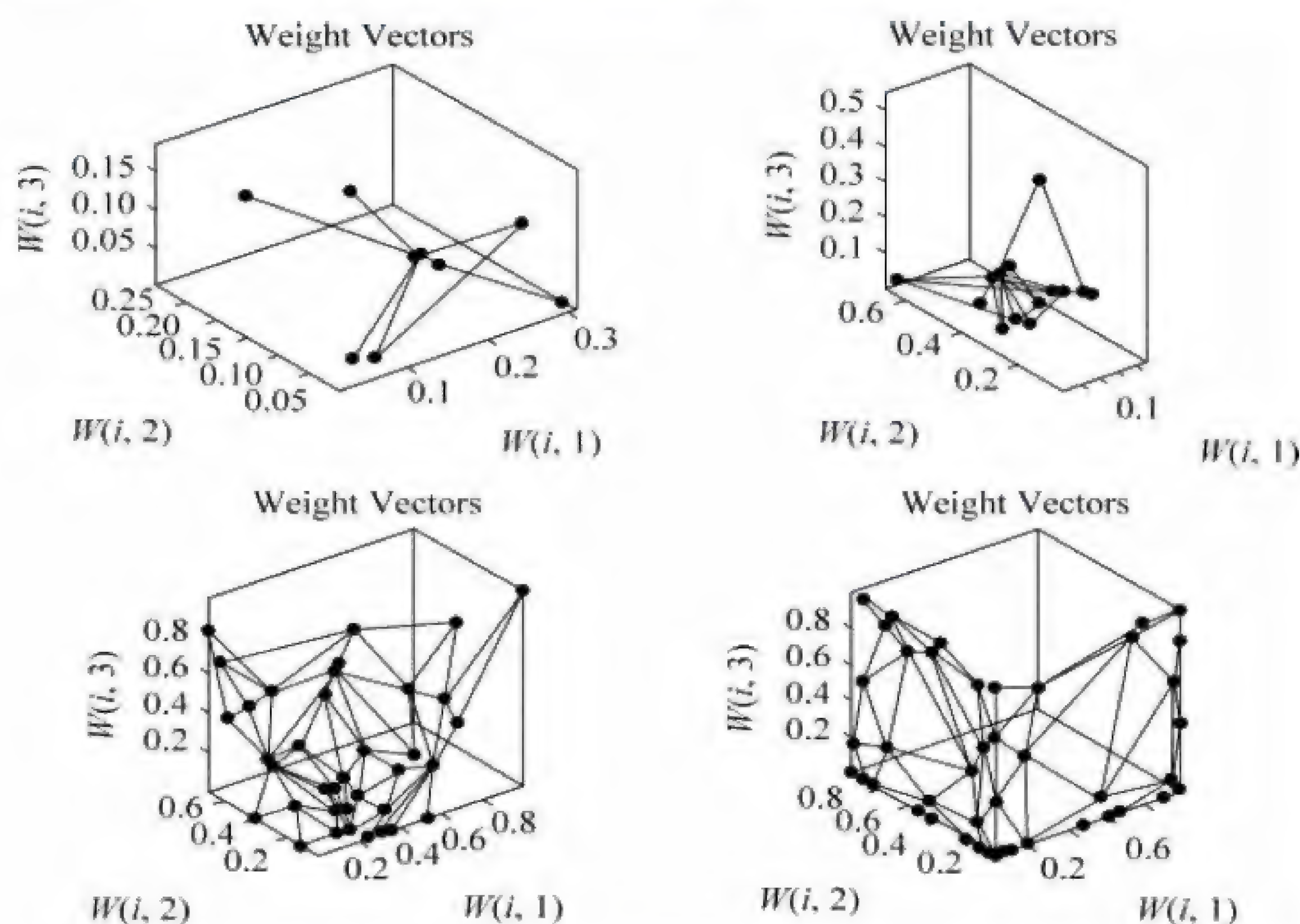


图 16-19　不同神经网络训练步数得到权值的变化

其中，训练次数为 80 时，用输入样本仿真得到的输出为

```
>> yc

yc =

    7.0000   7.0000   57.0000   1.0000   5.0000   1.0000   63.0000   14.0000   7.0000
63.0000   14.0000   57.0000   2.0000
   62.0000   60.0000   7.0000   7.0000   36.0000   14.0000   36.0000   63.0000   62.0000
43.0000   63.0000   8.0000   29.0000
   56.0000   63.0000   57.0000   50.0000   14.0000   55.0000   14.0000   1.0000   56.0000
7.0000   1.0000   58.0000   13.0000
   35.0000   14.0000   1.0000   29.0000   63.0000   32.0000   61.0000   57.0000   35.0000
52.0000   57.0000   5.0000   56.0000
   -0.1740   0.2750   -0.7020   -0.5440   -0.0434   -0.0841   -0.0056   0.1756
 -0.2589   -0.4291   -0.6771   0.1466   0.2970
   -0.1700   -0.0749   -0.3165   -0.5390   -0.1495   0.0482   0.2582   0.9719
 -0.8962   -0.0953   0.1206   0.4155   0.5951
    0.8202   -0.6154   -0.3063   0.2186   -0.5114   0.8945   0.8178   -0.9186   0.4166
 -0.6989   -0.2148   0.6673   0.7005
    0.4085   -0.7570   -0.3076   0.1715   -0.5692   -0.6014   -0.1176   0.8829
 -0.5680   -0.3958   0.7287   -0.6556   -0.4577
    0.2571   -0.5656   -0.8770   0.7700   0.2015   -0.5578   -0.7294   -0.9041
 -0.9028   -0.4660   -0.1369   0.3233   -0.6034
   -0.7162   0.2372   0.1541   -0.2522   -0.9802   -0.7279   -0.0757   0.2032
 -0.4721   -0.7632   -0.0727   0.6494   -0.1555
   -0.7245   -0.8315   -0.7497   0.2643   -0.4316   -0.5737   0.2970   0.5667
```

```
 -0.9628   -0.2353   -0.9101   0.3883   0.3579
     0.8758  0.7755   -0.9877   0.6451   -0.2747   0.6770   -0.7715   0.2235   0.4196
0.2380   -0.8980   -0.5681   -0.8745
     0.7493   -0.9630   0.9730   0.8614   -0.8371   0.1499   -0.8916   -0.1223
 -0.3245   -0.3393   -0.7133   -0.9777   0.6827
```

16.3 神经网络的经典应用

在网络模型与算法研究的基础上，利用人工神经网络组成实际的应用系统，例如完成某种信号处理或模式识别的功能、构作专家系统、制成机器人、复杂系统控制等。

纵观当代新兴科学技术的发展历史，人类在征服宇宙空间、基本粒子和生命起源等科学技术领域的进程中历经了崎岖不平的道路。我们看到，探索人脑功能和神经网络的研究将伴随着克服重重困难而日新月异。

本章重点介绍神经网络的经典应用。

16.3.1 PID 神经网络控制

在控制系统中，PID 控制是历史最悠久，生命力最强的控制方式，具有直观、实现简单和鲁棒性能好等一系列优点。

在实际应用中，经常会将 PID 与神经元网络融合起来。二者融合首先需要将 PID 功能引入神经网络的神经元中构成 PID 神经元，按照 PID 神经元的控制规律的基本模式，用这些基本神经元构成新的神经元网络，并找到合理有效的学习与计算方法。

将常规 PID 控制同神经网络相结合是现代控制理论的一个发展趋势。

PID 控制器主要由 PID 控制器和被控对象所组成。而 PID 控制器则由比例、积分、微分三个环节组成。它的数学描述为

$$u(t) = K_p\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)\mathrm{d}\tau + T_D\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right]$$

式中，K_p 为比例系数；

T_i 为积分时间常数；

T_d 为微分时间常数。

神经网络 PID 控制是神经网络应用于 PID 控制并与传统 PID 控制器相结合而产生的一种改进型控制方法，是对传统的 PID 控制的一种改进和优化。

单神经元构造 PID 控制器网络的输入为

$$X_1(k) = e(k)$$

$$X_2(k) = \sum_{j=0}^{k}(j)$$

$$X_3(k) = \Delta e(k) = e(k) - e(k-1)$$

PID 控制器网络网络的输出为

$$u(k) = W_1X_1(k) + W_2X_2(k) + W_3X_3(k)$$

式中,W_i 为控制器的加权系数,相当于 PID 控制器中的比例、积分、微分系数—K_p,K_I,K_D,但与传统的 PID 控制器不同的是参数 W_i 可以进行在线修正。通过不断调整 W_i 使之达到最优值 W^*,从而达到改善控制系统的控制性能的目的。

【例 16-6】 在 MATLAB 中,编写代码实现 PID 神经元网络控制系统,要求三个控制变量。

解:PID 神经网络控制器的 MATLAB 代码如下:

```
clear all
clc
%初始化
rate1 = 0.004;
rate2 = 0.007;
rate3 = 0.009;
k = 0.47;
K = 3;
y_1 = zeros(3,1);
y_2 = y_1;
y_3 = y_2;
u_1 = zeros(3,1);
u_2 = u_1;
u_3 = u_2;
h1i = zeros(3,1);
h1i_1 = h1i;                    %第一个控制量
h2i = zeros(3,1);
h2i_1 = h2i;                    %第二个控制量
h3i = zeros(3,1);
h3i_1 = h3i;                    %第三个控制量
x1i = zeros(3,1);
x2i = x1i;
x3i = x2i;
%隐含层
x1i_1 = x1i;
x2i_1 = x2i;
x3i_1 = x3i;
%初始化
k0 = 0.045;
%第一层权值
w11 = k0 * rand(3,2);
w11_1 = w11;
w11_2 = w11_1;
w12 = k0 * rand(3,2);
w12_1 = w12;
w12_2 = w12_1;
w13 = k0 * rand(3,2);
w13_1 = w13;
w13_2 = w13_1;
%第二层权值
w21 = k0 * rand(1,9);
```

```
w21_1 = w21;
w21_2 = w21_1;
w22 = k0 * rand(1,9);
w22_1 = w22;
w22_2 = w22_1;
w23 = k0 * rand(1,9);
w23_1 = w23;
w23_2 = w23_1;

%限定值
ynmax = 1;
ynmin = -1; %系统输出值限定
xpmax = 1;
xpmin = -1; %P节点输出限定
qimax = 1;
qimin = -1; %I节点输出限定
qdmax = 1;
qdmin = -1; %D节点输出限定
uhmax = 1;
uhmin = -1; %输出结果限定
%网络迭代优化
for k = 1:1:150
    %系统输出
    y1(k) = (0.4 * y_1(1) + u_1(1)/(1 + u_1(1)^2) + 0.2 * u_1(1)^3 + 0.5 * u_1(2)) + 0.3 * y_1
(2);
    y2(k) = (0.2 * y_1(2) + u_1(2)/(1 + u_1(2)^2) + 0.4 * u_1(2)^3 + 0.2 * u_1(1)) + 0.3 * y_1
(3);
    y3(k) = (0.3 * y_1(3) + u_1(3)/(1 + u_1(3)^2) + 0.4 * u_1(3)^3 + 0.4 * u_1(2)) + 0.3 * y_1
(1);
    r1(k) = 0.7;
    r2(k) = 0.5;
    r3(k) = 0.9;
    %限制系统输出
    yn = [y1(k),y2(k),y3(k)];
    yn(find(yn > ynmax)) = ynmax;
    yn(find(yn < ynmin)) = ynmin;
    %输入层
    x1o = [r1(k);yn(1)];x2o = [r2(k);yn(2)];x3o = [r3(k);yn(3)];
    %隐含层
    x1i = w11 * x1o;
    x2i = w12 * x2o;
    x3i = w13 * x3o;

    %比例神经元P计算
    xp = [x1i(1),x2i(1),x3i(1)];
    xp(find(xp > xpmax)) = xpmax;
```

```
xp(find(xp<xpmin))=xpmin;
qp=xp;
h1i(1)=qp(1);
h2i(1)=qp(2);
h3i(1)=qp(3);

%积分神经元I计算
xi=[x1i(2),x2i(2),x3i(2)];
qi=[0,0,0];
qi_1=[h1i(2),h2i(2),h3i(2)];
qi=qi_1+xi;
qi(find(qi>qimax))=qimax;
qi(find(qi<qimin))=qimin;
h1i(2)=qi(1);
h2i(2)=qi(2);
h3i(2)=qi(3);

%微分神经元D计算
xd=[x1i(3),x2i(3),x3i(3)];
qd=[0 0 0];
xd_1=[x1i_1(3),x2i_1(3),x3i_1(3)];
qd=xd-xd_1;
qd(find(qd>qdmax))=qdmax;
qd(find(qd<qdmin))=qdmin;
h1i(3)=qd(1);
h2i(3)=qd(2);
h3i(3)=qd(3);

%输出层计算
wo=[w21;w22;w23];
qo=[h1i',h2i',h3i'];
qo=qo';
uh=wo*qo;
uh(find(uh>uhmax))=uhmax;
uh(find(uh<uhmin))=uhmin;
u1(k)=uh(1);
u2(k)=uh(2);
u3(k)=uh(3);
%计算误差
error=[r1(k)-y1(k);r2(k)-y2(k);r3(k)-y3(k)];
error1(k)=error(1);
error2(k)=error(2);
error3(k)=error(3);
J(k)=0.5*(error(1)^2+error(2)^2+error(3)^2);%调整大小
ypc=[y1(k)-y_1(1);y2(k)-y_1(2);y3(k)-y_1(3)];
```

```
uhc = [u_1(1) - u_2(1);u_1(2) - u_2(2);u_1(3) - u_2(3)];
% 隐含层和输出层权值调整
Sig1 = sign(ypc./(uhc(1) + 0.00001));
dw21 = sum(error. * Sig1) * qo';
w21 = w21 + rate2 * dw21 + rate3 * (w21_1 - w21_2);
Sig2 = sign(ypc./(uh(2) + 0.00001));
dw22 = sum(error. * Sig2) * qo';
w22 = w22 + rate2 * dw22 + rate3 * (w22_1 - w21_2);
Sig3 = sign(ypc./(uh(3) + 0.00001));
dw23 = sum(error. * Sig3) * qo';
w23 = w23 + rate2 * dw23 + rate3 * (w23_1 - w23_2);

% 调整输入层和隐含层权值
delta2 = zeros(3,3);
wshi = [w21;w22;w23];
for t = 1:1:3
    delta2(1:3,t) = error(1:3). * sign(ypc(1:3)./(uhc(t) + 0.0000001));
end
for j = 1:1:3
    sgn(j) = sign((h1i(j) - h1i_1(j))/(x1i(j) - x1i_1(j) + 0.0001));
end

 s1 = sgn' * [r1(k),y1(k)];
 wshi2_1 = wshi(1:3,1:3);
 alter = zeros(3,1);
 dws1 = zeros(3,2);
 for j = 1:1:3
     for p = 1:1:3
         alter(j) = alter(j) + delta2(p,:) * wshi2_1(:,j);
     end
 end

 for p = 1:1:3
     dws1(p,:) = alter(p) * s1(p,:);
 end
 w11 = w11 + rate1 * dws1 + rate3 * (w11_1 - w11_2);

 % 调整权值
for j = 1:1:3
    sgn(j) = sign((h2i(j) - h2i_1(j))/(x2i(j) - x2i_1(j) + 0.0000001));
end
s2 = sgn' * [r2(k),y2(k)];
wshi2_2 = wshi(:,4:6);
alter2 = zeros(3,1);
dws2 = zeros(3,2);
```

```
    for j = 1:1:3
        for p = 1:1:3
            alter2(j) = alter2(j) + delta2(p,:) * wshi2_2(:,j);
        end
    end
    for p = 1:1:3
        dws2(p,:) = alter2(p) * s2(p,:);
    end
    w12 = w12 + rate1 * dws2 + rate3 * (w12_1 - w12_2);

    for j = 1:1:3
        sgn(j) = sign((h3i(j) - h3i_1(j))/(x3i(j) - x3i_1(j) + 0.0000001));
    end
    s3 = sgn' * [r3(k),y3(k)];
    wshi2_3 = wshi(:,7:9);
    alter3 = zeros(3,1);
    dws3 = zeros(3,2);
    for j = 1:1:3
        for p = 1:1:3
            alter3(j) = (alter3(j) + delta2(p,:) * wshi2_3(:,j));
        end
    end
    for p = 1:1:3
        dws3(p,:) = alter2(p) * s3(p,:);
    end
    w13 = w13 + rate1 * dws3 + rate3 * (w13_1 - w13_2);

    % 参数更新
    u_3 = u_2;
    u_2 = u_1;
    u_1 = uh;
    y_2 = y_1;
    y_1 = yn;
    h1i_1 = h1i;
    h2i_1 = h2i;
    h3i_1 = h3i;
    x1i_1 = x1i;
    x2i_1 = x2i;
    x3i_1 = x3i;
    w11_1 = w11;
    w11_2 = w11_1;
    w12_1 = w12;
    w12_2 = w12_1;
    w13_1 = w13;
    w13_2 = w13_1;
```

```
    % 第二层权值
    w21_1 = w21;
    w21_2 = w21_1;
    w22_1 = w22;
    w22_2 = w22_1;
    w23_1 = w23;
    w23_2 = w23_1;
end

% 结果分析
time = 0.001 * (1:k);
figure(1)
subplot(3,1,1)
plot(time,r1,'r-',time,y1,'b-');
title('PID 神经网络控制','fontsize',12);
ylabel('控制量 1','fontsize',12);
subplot(3,1,2)
plot(time,r2,'r-',time,y2,'b-');
ylabel('控制量 2','fontsize',12);
subplot(3,1,3)
plot(time,r3,'r-',time,y3,'b-');
xlabel('时间/秒','fontsize',12);
ylabel('控制量 3','fontsize',12);

figure(2)
plot(time,u1,'r-',time,u2,'g-',time,u3,'b');
title('对象的控制输入');
xlabel('时间');
ylabel('被控量');
legend('u1','u2','u3');
grid on

figure(3)
plot(time,J,'r-');
axis([0,0.3,0,2]);
grid on
title('控制误差曲线','fontsize',12);
xlabel('时间','fontsize',12);
ylabel('控制误差','fontsize',12);
```

运行代码后，得到 PID 控制器的控制效果如图 16-20 所示。

对象的控制输入量如图 16-21 所示。

运行代码得到的控制器误差如图 16-22 所示。

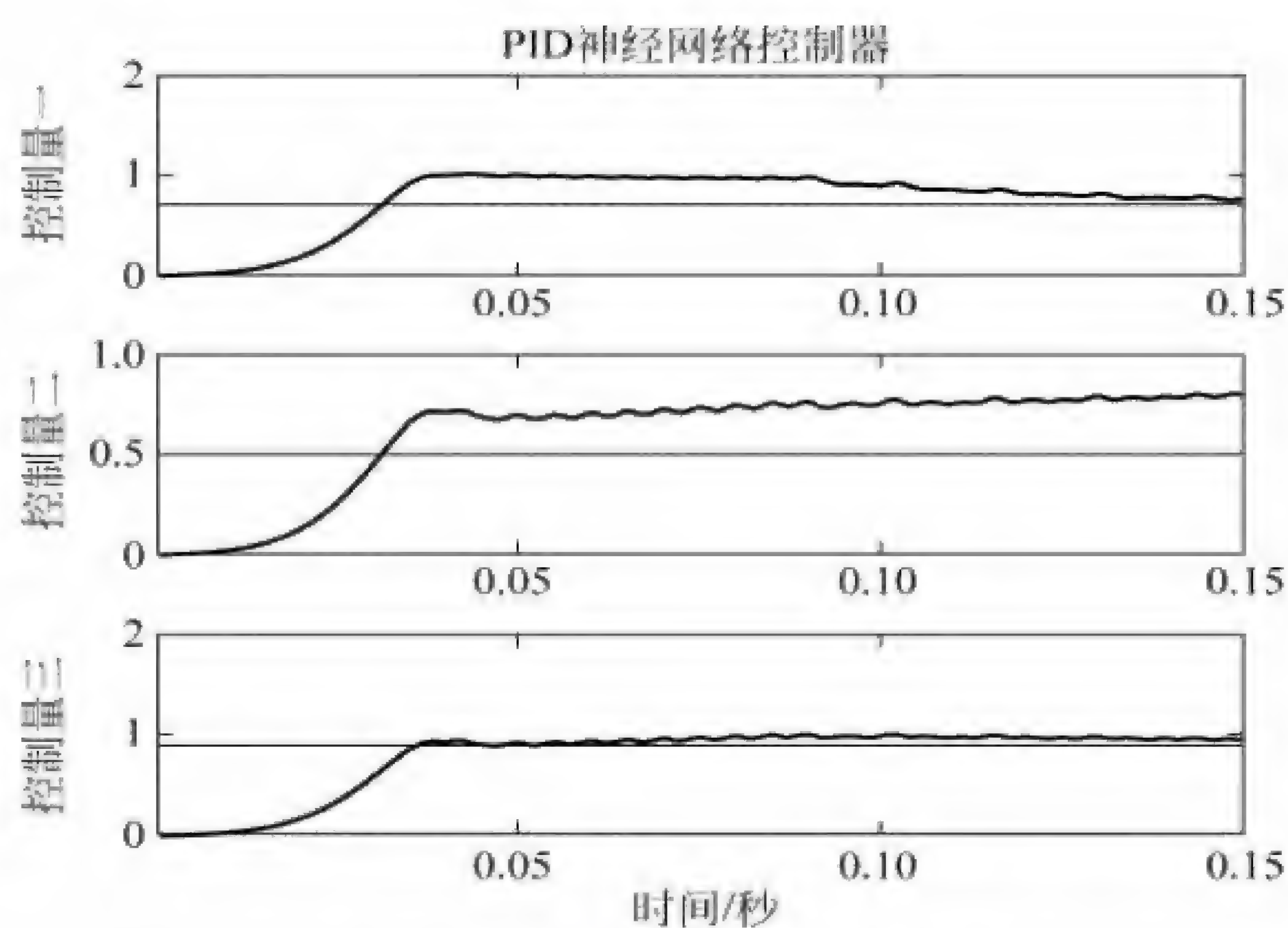


图 16-20　可控制器控制效果图

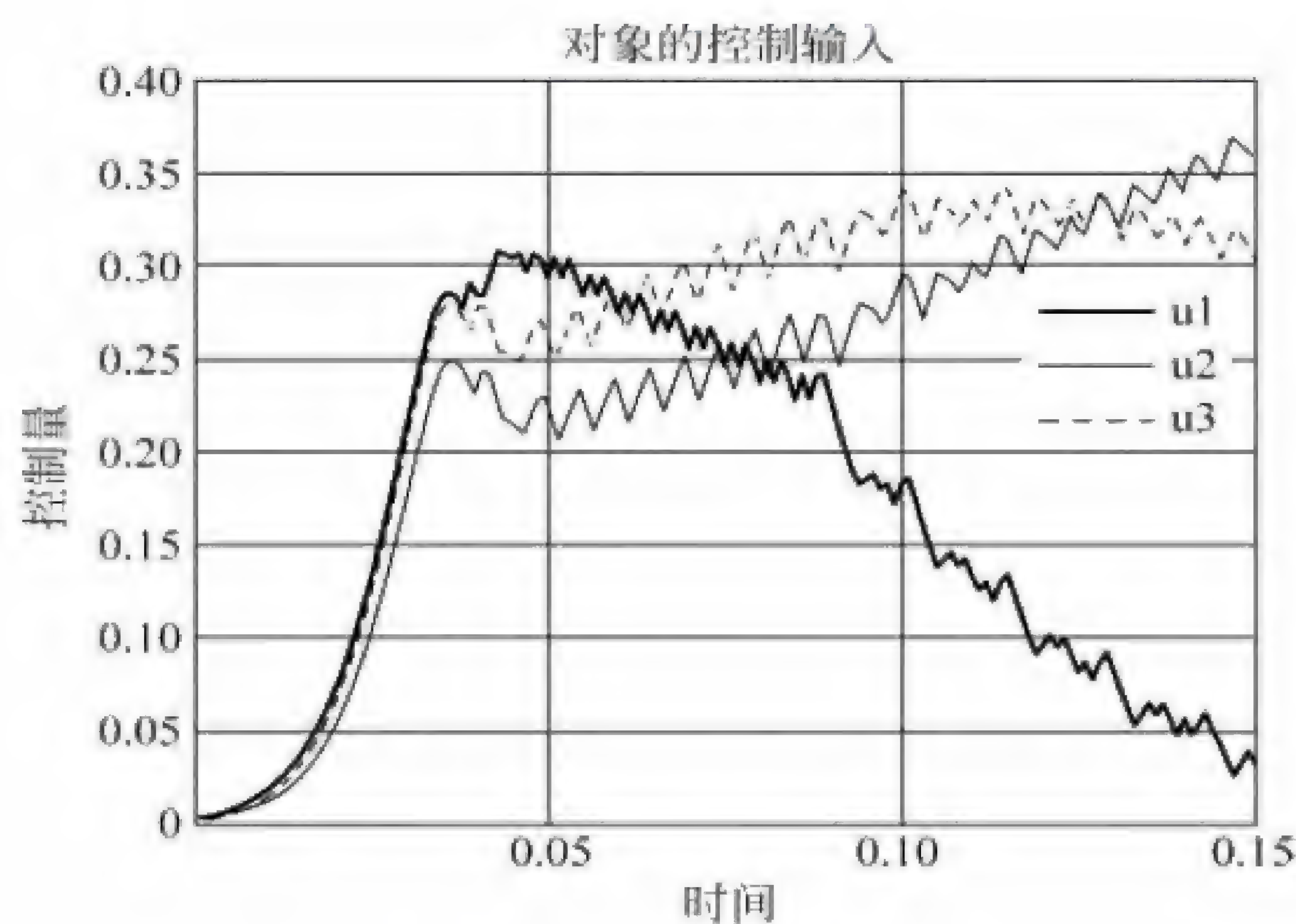


图 16-21　控制器的误差

16.3.2　模糊神经网络在函数逼近中的应用

函数逼近问题是在选定的一类函数中寻找某个函数 m，近似表示已知函数 n，并求出用 m 近似表示 n 产生的误差。

模糊神经网络就是模糊理论同神经网络相结合的产物，它汇集了神经网络与模糊理论的优点，集学习、联想、识别、信息处理于一体，在处理非线性、模糊性等问题上有很大的优越性，在函数逼近方面存在巨大的潜力。

【例 16-7】　已知目标函数为 $y=\frac{1}{4}(\pi x_1^2)\cos(\pi x_2^2)$，编写 MATLAB 代码实现模糊神

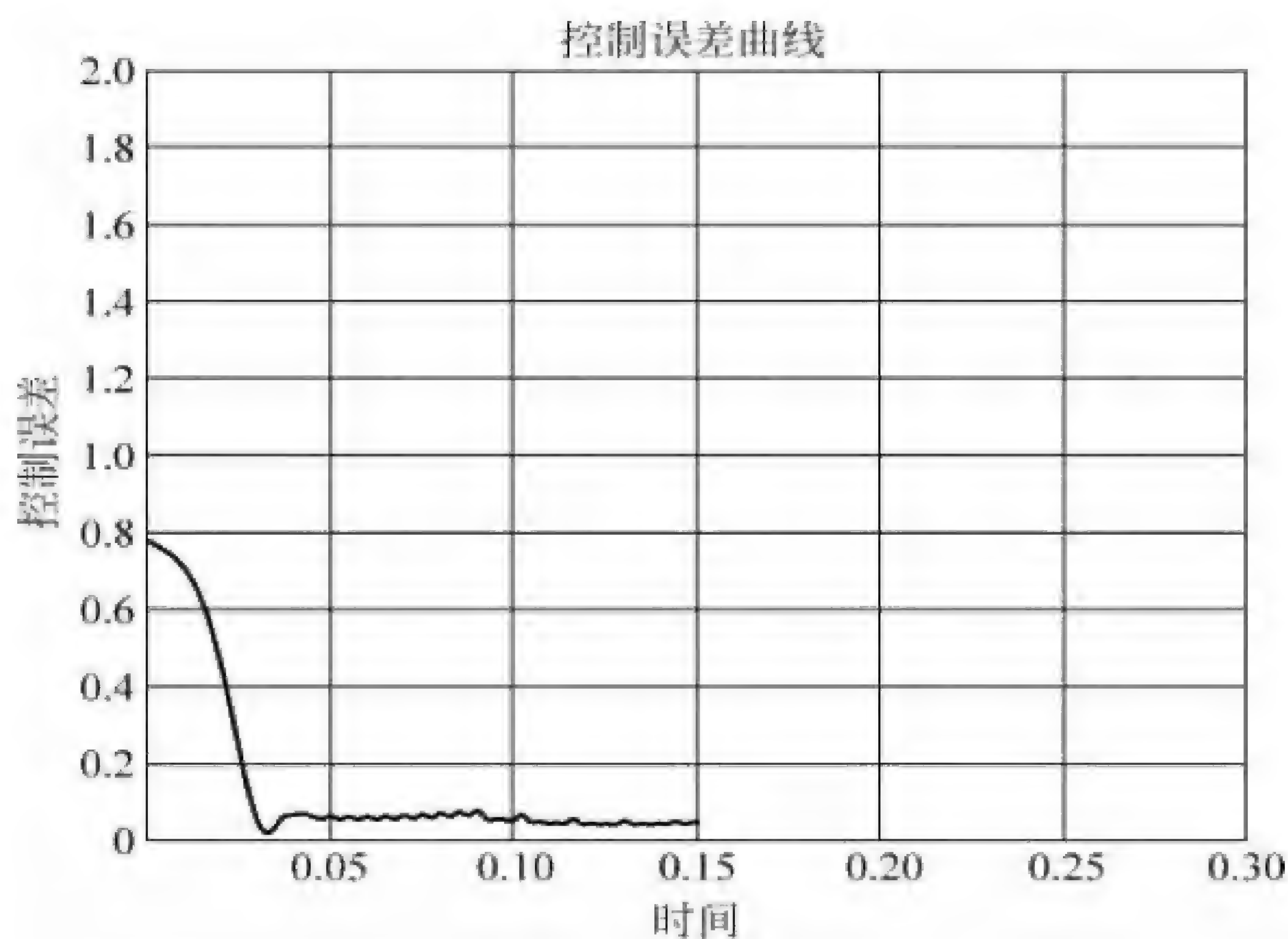


图 16-22 控制器的误差

经网络函数分别在区间$\begin{cases} x_1 \in [0,1], & x_2 \in [0,1] \\ x_1 \in [-0.5,1], & x_2 \in [-1,0.5] \\ x_1 \in [-2,2], & x_2 \in [-2,2] \end{cases}$逼近目标函数。

解：(1) 当区间为 $x_1 \in [0,1]$，$x_2 \in [0,1]$时，编写 MATLAB 的代码如下：

```
clear all
clc
[x1,x2] = meshgrid(0:0.05:1,0:0.05:1);
y = 0.2 * (pi * (x1.^2)). * cos(pi * x2.^2);          % 求得函数输出值
x11 = reshape(x1,441,1);                               % 将输入变量变为列向量
x12 = reshape(x2,441,1);
y1 = reshape(y,441,1);                                 % 将输出变量变为列向量
trnData = [x11(1:2:441) x12(1:2:441) y1(1:2:441)];     % 构造训练数据
chkData = [x11 x12 y1];                                % 构造测试数据
numMFs = 6;                                            % 定义隶属函数个数
mfType = 'gbellmf';                                    % 定义隶属函数类型
epoch_n = 30;                                          % 定义训练次数
in_fisMat = genfis1(trnData,numMFs,mfType);            % 由训练数据直接生成模糊推理系统
out_fisMat = anfis(trnData,in_fisMat,30);              % 训练模糊系统
y11 = evalfis(chkData(:,1:2),out_fisMat);              % 用测试数据测试系统
x111 = reshape(x11,21,21);
x112 = reshape(x12,21,21);
y111 = reshape(y11,21,21);
figure(1)
subplot(2,2,1),
mesh(x1,x2,y);
title('期望输出');
subplot(2,2,2),
mesh(x111,x112,y111);
title('实际输出');
```

```
subplot(2,2,3),
mesh(x1,x2,(y-y111));
title('误差');
[x,mf]=plotmf(in_fisMat,'input',1);
[x,mf1]=plotmf(out_fisMat,'input',1);
subplot(2,2,4),
plot(x,mf,'r-',x,mf1,'k--');
title('隶属度函数变化');
figure(2)
gensurf(out_fisMat)
title('推理输入输出关系图');
xlabel('输入x1');
ylabel('输入x2');
zlabel('输出y');
```

运行后得到如下结果：

```
ANFIS info:
    Number of nodes: 101
    Number of linear parameters: 108
    Number of nonlinear parameters: 36
    Total number of parameters: 144
    Number of training data pairs: 221
    Number of checking data pairs: 0
    Number of fuzzy rules: 36

Start training ANFIS ...

   1    0.00215076
   2    0.00194422
   3    0.00174838
   4    0.00156094
   5    0.00138166
Step size increases to 0.011000 after epoch 5.
   6    0.00120957
   7    0.00102282
   8    0.000842614
   9    0.000998671
  10    0.000794749
  11    0.000828458
  12    0.000751107
Step size decreases to 0.009900 after epoch 12.
  13    0.000701162
  14    0.000689433
  15    0.000612648
Step size increases to 0.010890 after epoch 15.
  16    0.000645093
  17    0.000576041
  18    0.000609313
```

```
  19    0.000501842
Step size decreases to 0.009801 after epoch 19.
  20    0.000571735
  21    0.000420345
  22    0.000472761
  23    0.000434854
Step size decreases to 0.008821 after epoch 23.
  24    0.000455408
  25    0.00035057
  26    0.000434034
  27    0.000323531
Step size decreases to 0.007939 after epoch 27.
  28    0.000415174
  29    0.000280044
  30    0.000360497
Designated epoch number reached --> ANFIS training completed at epoch 30.
```

运行代码后，得到结果如图 16-23 所示，推理输入和输出关系图如图 16-24。

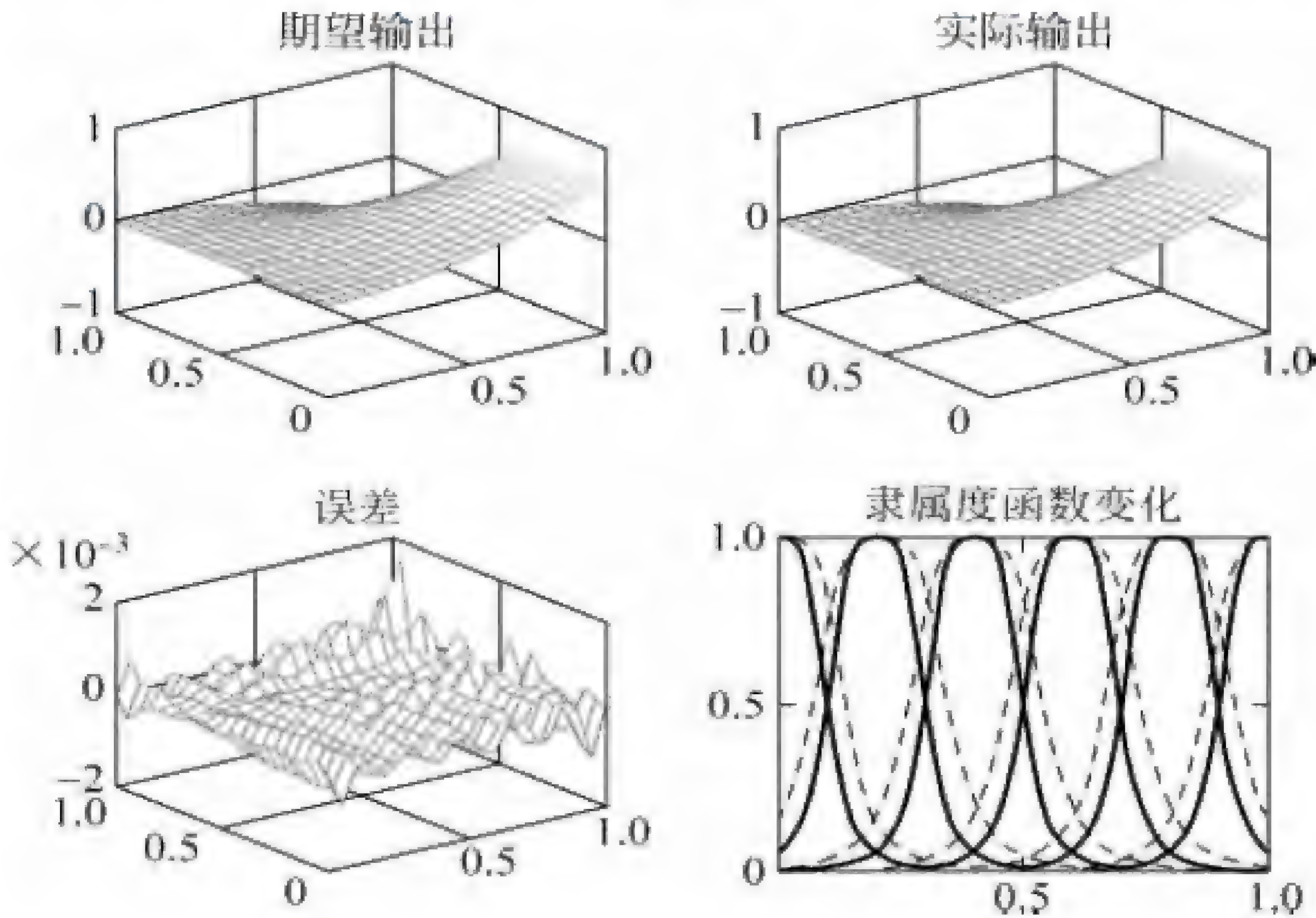


图 16-23　代码运行结果图

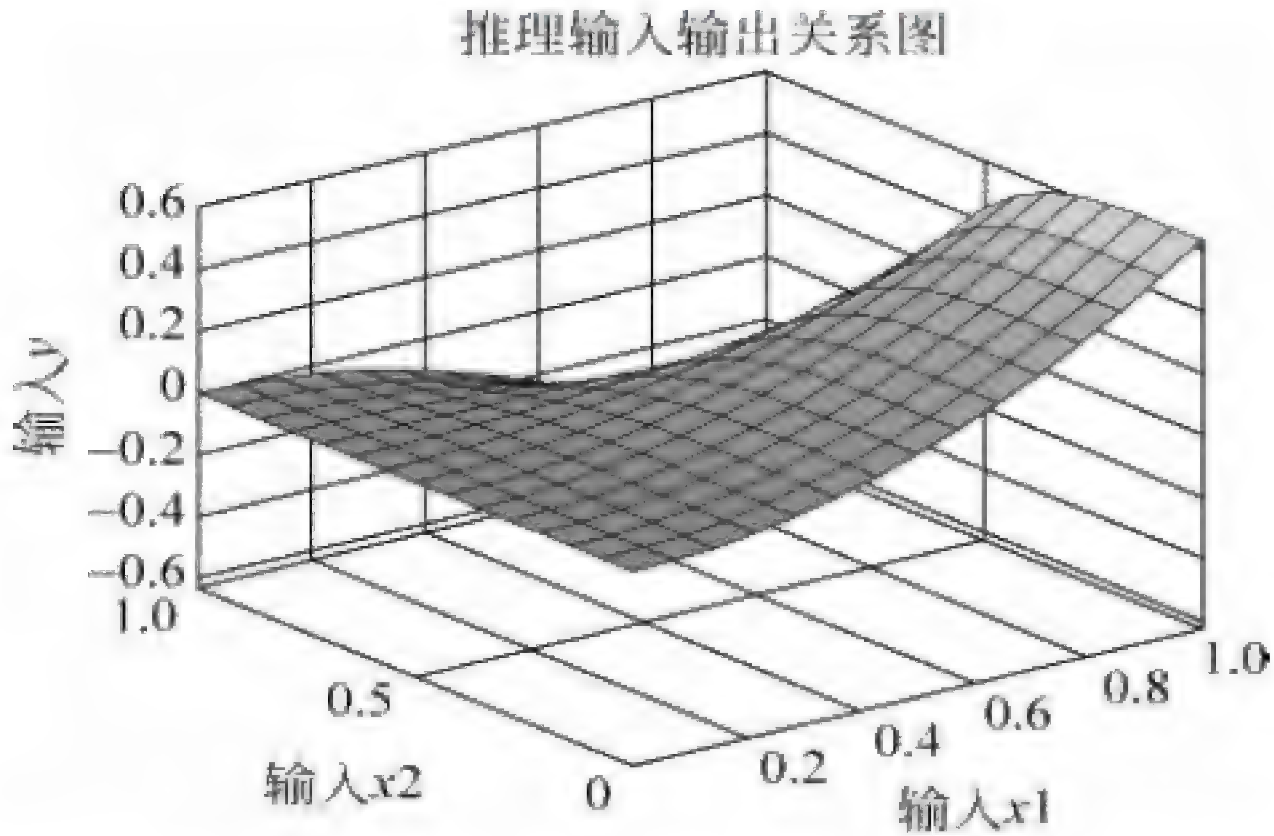


图 16-24　推理输入和输出关系图

(2) 当 $x_1 \in [-0.5,1]$, $x_2 \in [-1,0.5]$时,编写 MATLAB 的代码如下:

```
clear all
clc
[x1,x2] = meshgrid( - 0.5:0.05:1, - 1:0.05:0.5);      % 将输入空间划分为 31 * 31 个网格点
y = 0.2 * (pi * (x1.^2)). * cos(pi * x2.^2);          % 求得函数输出值
x11 = reshape(x1,961,1);                               % 将输入变量变为列向量
x12 = reshape(x2,961,1);                               % 将输入变量变为列向量
y1 = reshape(y,961,1);                                 % 将输出变量变为列向量
trnData = [x11(1:2:961) x12(1:2:961) y1(1:2:961)];     % 构造训练数据
chkData = [x11 x12 y1];                                % 构造测试数据
numMFs = 4;                                            % 定义隶属函数个数
mfType = 'gbellmf';                                    % 定义隶属函数类型
epoch_n = 40                                           % 设置训练次数
in_fisMat = genfis1(trnData,numMFs,mfType);            % 采用 genfis1 函数由训练数据直接
                                                         生成模糊推理系统
out_fisMat = anfis(trnData,in_fisMat,40);              % 训练模糊系统
y11 = evalfis(chkData(:,1:2),out_fisMat);              % 用测试数据测试系统
x111 = reshape(x11,31,31);
x112 = reshape(x12,31,31);
y111 = reshape(y11,31,31);
figure(1)
subplot(2,2,1),
mesh(x1,x2,y);
title('期望输出');
subplot(2,2,2),
mesh(x111,x112,y111);
title('实际输出');
subplot(2,2,3),
mesh(x1,x2,(y - y111));
title('误差');
[x,mf] = plotmf(in_fisMat,'input',1);
[x,mf1] = plotmf(out_fisMat,'input',1);
subplot(2,2,4),
plot(x,mf,'r - ',x,mf1,'k -- ');
title('隶属度函数变化');
figure(2)
gensurf(out_fisMat)
title('推理输入输出关系图');
xlabel('输入 x1');
ylabel('输入 x2');
zlabel('输出 y');
```

运行代码后得到如下结果:

```
ANFIS info:
    Number of nodes: 53
    Number of linear parameters: 48
```

```
    Number of nonlinear parameters: 24
    Total number of parameters: 72
    Number of training data pairs: 481
    Number of checking data pairs: 0
    Number of fuzzy rules: 16


Start training ANFIS ...

   1    0.00767841
   2    0.00733078
   3    0.00699163
   4    0.00665565
   5    0.0063187
Step size increases to 0.011000 after epoch 5.
   6    0.00597808
   7    0.00559829
   8    0.00521619
   9    0.00484197
Step size increases to 0.012100 after epoch 9.
  10    0.00449545
  11    0.00417039
  12    0.0038745
  13    0.00358538
Step size increases to 0.013310 after epoch 13.
  14    0.00330713
  15    0.00301713
  16    0.00274759
  17    0.00250295
Step size increases to 0.014641 after epoch 17.
  18    0.00229804
  19    0.00223414
  20    0.00217776
  21    0.00213956
Step size increases to 0.016105 after epoch 21.
  22    0.00206527
  23    0.00211321
  24    0.00198189
  25    0.00204968
  26    0.0019156
Step size decreases to 0.014495 after epoch 26.
  27    0.00199749
  28    0.0018004
  29    0.00193928
  30    0.00175525
Step size decreases to 0.013045 after epoch 30.
  31    0.00189146
  32    0.00167651
  33    0.0018451
  34    0.00165391
```

```
Step size decreases to 0.011741 after epoch 34.
   35    0.00181522
   36    0.00160446
   37    0.00177368
   38    0.00158601
Step size decreases to 0.010567 after epoch 38.
   39    0.00174304
   40    0.00154876

Designated epoch number reached --> ANFIS training completed at epoch 40.
```

运行代码后，得到结果如图 16-25 所示，推理输入和输出关系图如图 16-26。

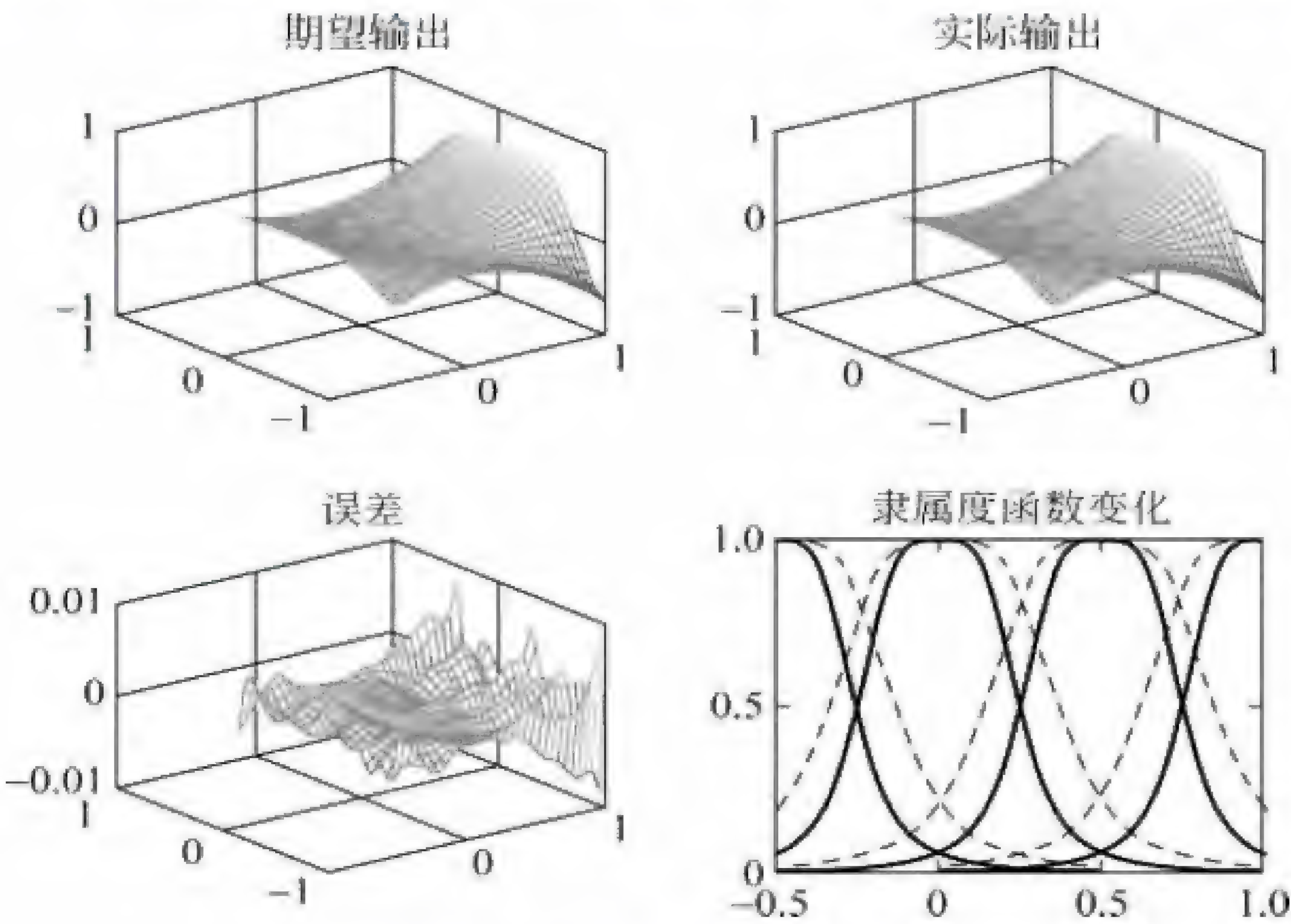


图 16-25　代码运行结果图

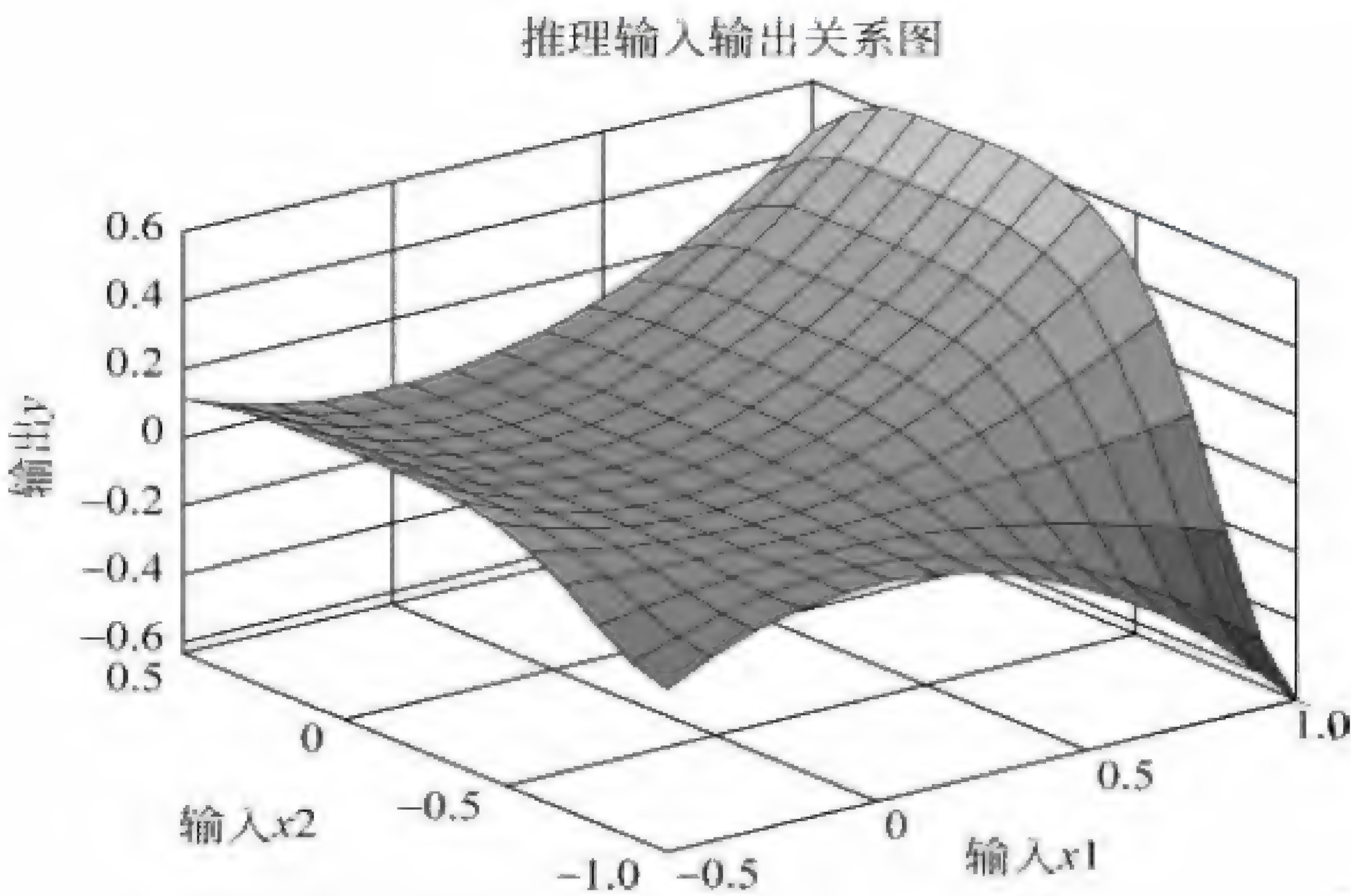


图 16-26　推理输入和输出关系图

(3) 当 $x_1 \in [-2,2]$，$x_2 \in [-2,2]$时，编写 MATLAB 代码逼近目标函数如下：

```
clear all
clc
[x1,x2] = meshgrid( - 2:0.05:2, - 2:0.05:2);
y = 0.2 * (pi * (x1.^2)). * cos(pi * x2.^2);          % 求得函数输出值
x11 = reshape(x1,6561,1);
x12 = reshape(x2,6561,1);
y1 = reshape(y,6561,1);
trnData = [x11(1:2:6561) x12(1:2:6561) y1(1:2:6561)];
chkData = [x11 x12 y1];
numMFs = 4;                                           % 定义隶属函数个数
mfType = 'gbellmf';                                   % 定义隶属函数类型
epoch_n = 50;
in_fisMat = genfis1(trnData,numMFs,mfType);
out_fisMat = anfis(trnData,in_fisMat,50);
y11 = evalfis(chkData(:,1:2),out_fisMat);
x111 = reshape(x11,81,81);
x112 = reshape(x12,81,81);
y111 = reshape(y11,81,81);
figure(1)
subplot(2,2,1),
mesh(x1,x2,y);
title('期望输出');
subplot(2,2,2),
mesh(x111,x112,y111);
title('实际输出');
subplot(2,2,3),
mesh(x1,x2,(y - y111));
title('误差');
[x,mf] = plotmf(in_fisMat,'input',1);
[x,mf1] = plotmf(out_fisMat,'input',1);
subplot(2,2,4),
plot(x,mf,'r - ',x,mf1,'k -- ');
title('隶属度函数变化');
figure(2)
gensurf(out_fisMat)
title('推理输入输出关系图');
xlabel('输入 x1');
ylabel('输入 x2');
zlabel('输出 y');
```

运行代码后得到如下结果：

```
ANFIS info:
    Number of nodes: 53
    Number of linear parameters: 48
    Number of nonlinear parameters: 24
```

```
    Total number of parameters: 72
    Number of training data pairs: 431
    Number of checking data pairs: 0
    Number of fuzzy rules: 16


Start training ANFIS ...

   1    0.0196481
   2    0.0190324
   3    0.018409
   4    0.0177785
   5    0.0171409
Step size increases to 0.011000 after epoch 5.
   6    0.0164965
   7    0.0157797
   8    0.0150542
   9    0.0143193
Step size increases to 0.012100 after epoch 9.
  10    0.0135741
  11    0.0127415
  12    0.0118959
  13    0.0110426
Step size increases to 0.013310 after epoch 13.
  14    0.0101999
  15    0.00934149
  16    0.00866895
  17    0.00818413
Step size increases to 0.014641 after epoch 17.
  18    0.00770135
  19    0.00717948
  20    0.00668722
  21    0.00623409
Step size increases to 0.016105 after epoch 21.
  22    0.00588064
  23    0.00558347
  24    0.0054074
  25    0.00510844
Step size increases to 0.017716 after epoch 25.
  26    0.00502308
  27    0.00482777
  28    0.00476525
  29    0.0045367
Step size increases to 0.019487 after epoch 29.
  30    0.00455495

Designated epoch number reached --> ANFIS training completed at epoch 30.
```

运行代码后,得到结果如图 16-27 所示,推理输入和输出关系图如图 16-28 所示。

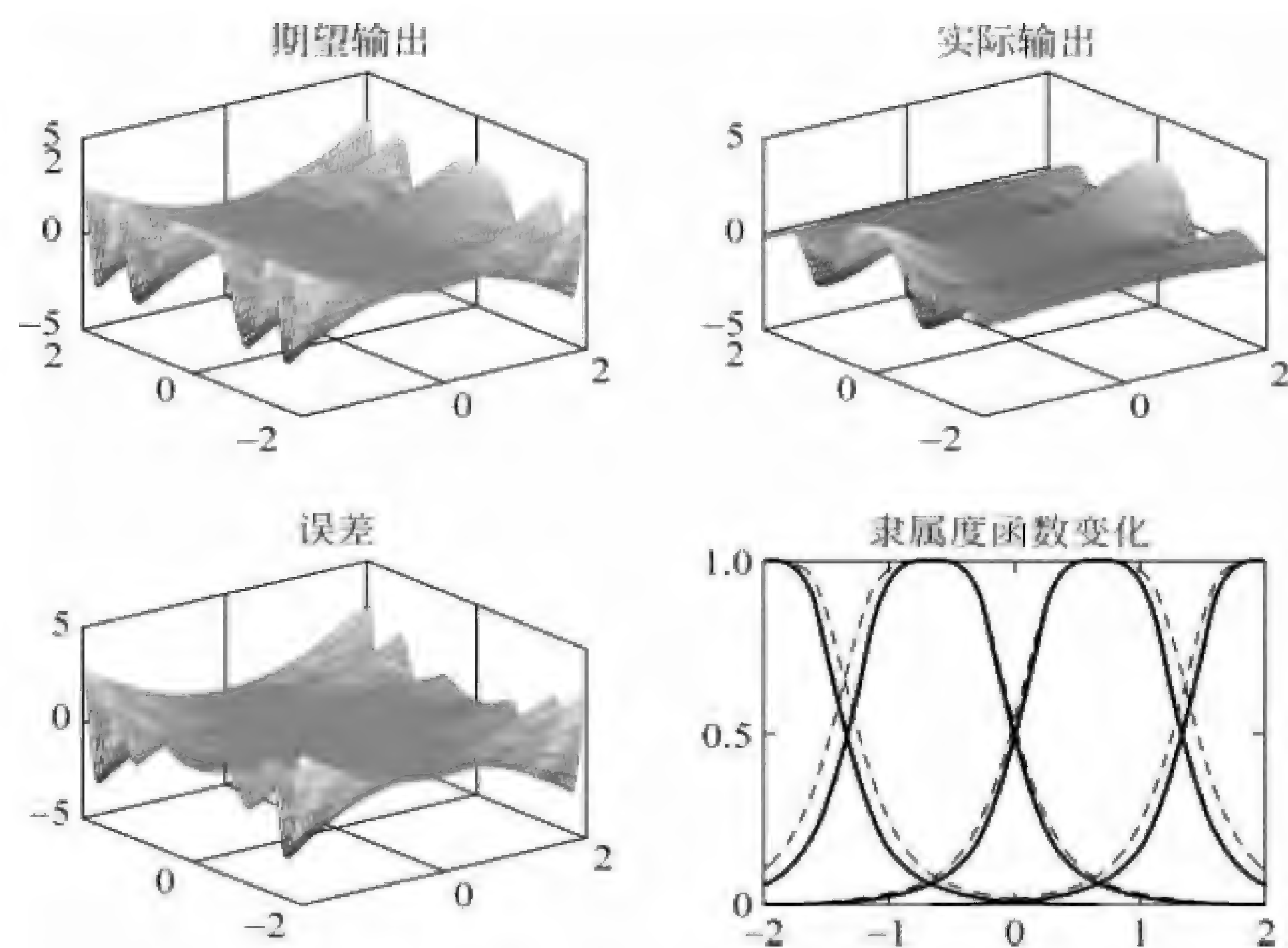


图 16-27　代码运行结果图

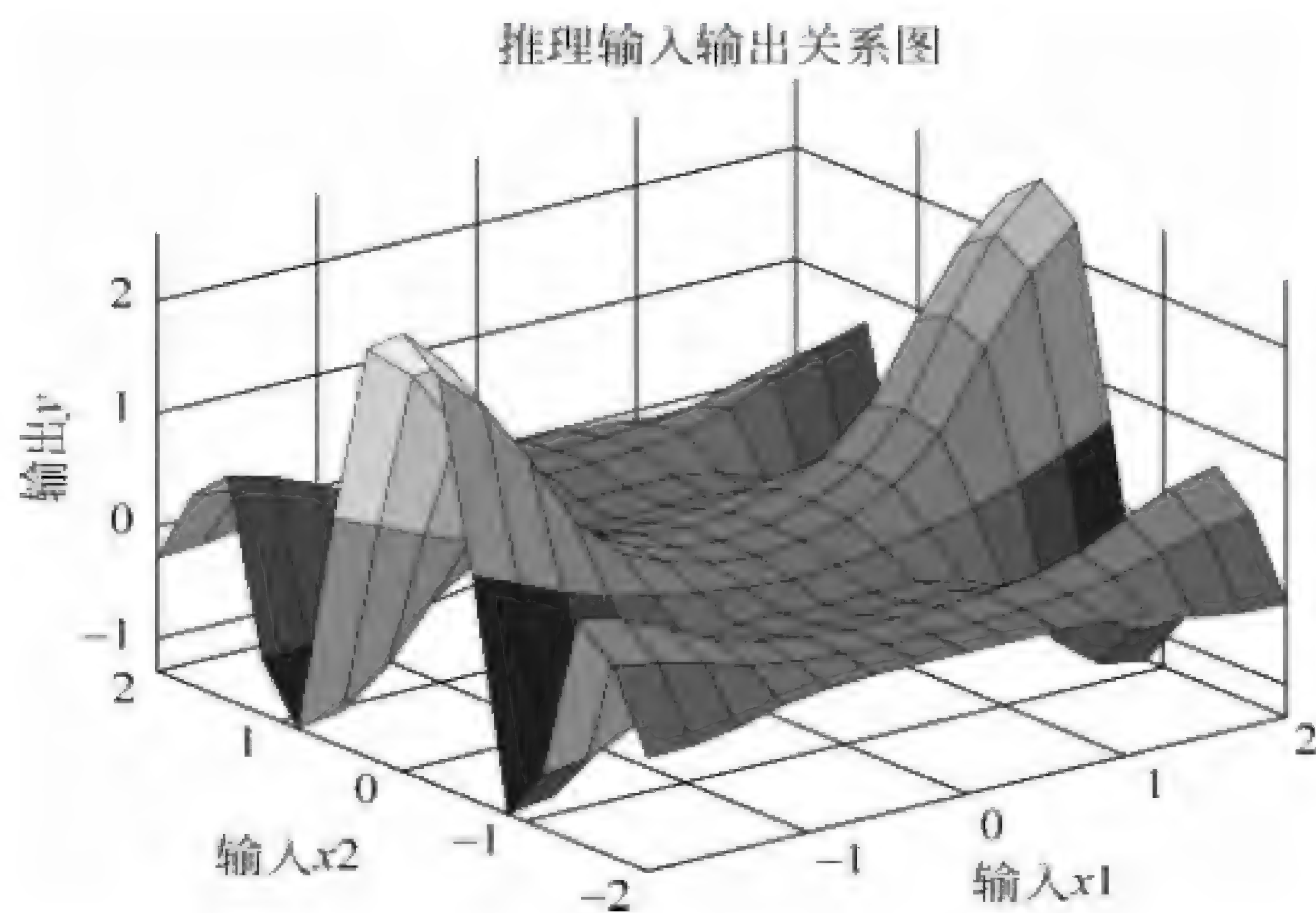
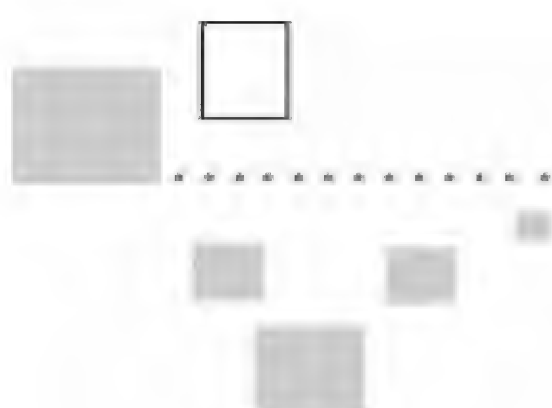


图 16-28　推理输入和输出关系图

本章小结

人工神经网络是模拟人思维的第二种方式。它是一个非线性动力学系统，其特色在于信息的分布式存储和并行协同处理。大量神经元构成的网络系统所能实现的行为是极其丰富多彩的。

本章首先介绍了神经网络的基本概念，然后对 MATLAB 神经网络工具箱做了重点介绍，最后详细介绍了几种神经网络的经典应用。



第四部分
MATLAB综合应用

第17章 分形维数应用与实现

被誉为大自然的几何学的分形(fractal)理论，是现代数学的一个新分支，但其本质是一种新的世界观和方法论。分维反映了复杂形体占有空间的有效性，它是复杂形体不规则性的量度。

学习目标：

(1) 了解分形维数概念；

(2) 掌握MATLAB在二维分形维数的应用；

(3) 熟练掌握MATLAB在分形插值算法中的应用。

17.1 分形维数概述

分形几何的概念是美籍法国数学家曼德布罗特(B. B. Mandelbrot)于1975年首先提出的。1934年，贝塞考维奇(A. S. Besicovitch)更深刻地提出了豪斯道夫测度的性质和奇异集的分数维，他在豪斯道夫测度及其几何的研究领域中作出了主要贡献，从而产生了豪斯道夫-贝塞考维奇维数概念。

以后，这一领域的研究工作没有引起更多人的注意，先驱们的工作只是作为分析与拓扑学教科书中的范例而流传开来。

分形包括规则分形和无规则分形两种。规则分形是指可以由简单的迭代或是按一定规律所生成的分形，如Cantor集、Koch曲线、Sierpinski海绵等。这些分形图形具有严格的自相似性。

分形维数是分形几何理论及应用中最为重要的概念和内容，它是度量物体或分形体复杂性和不规则性的最重要的指标，是定量描述分形自相似性程度大小的参数。

整数维数是被包含在分数维数中的。相对于整数维数反映对象的静态特征，分数维数则表征的是对象动态的变化过程。将其扩展到自然界的动态行为和现象中，分数维数就是自然现象中由细小局部特征构成整体系统行为的相关性的一种表征，即对于一个对象，只有通过使用非整数数值的维数尺度去度量它，才能准确地反映其所具有的不规则性和复杂程度，那么这个非整数数值的维数就称为分形维数。

【例 17-1】 使用 MATLAB 代码实现 Koch 分形曲线：从一条直线段开始，将线段中间的三分之一部分用一个等边三角形的两边代替，形成如图 17-1 所示的山丘图形。在新的图形中，又将图中每一直线段中间的三分之一部分都用一个等边三角形的两条边代替，再次形成新的图形如此迭代，形成 Koch 分形曲线。

解：考虑 2 个点产生 5 个点的过程。在图 17-1 中，设 P_1 和 P_5 分别为原始直线段的两个端点，现需要在直线段的中间依次插入三个点 P_2、P_3、P_4。其中，P_2 位于线段三分之一处，P_4 位于线段三分之二处，P_3 点的位置可看成是由 P_4 点以 P_2 点为轴心，逆时针旋转 60°得到。

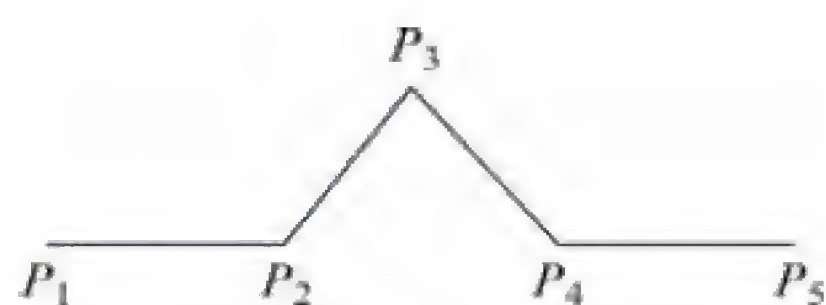


图 17-1　山丘图像

旋转由正交矩阵 $\boldsymbol{A}=\begin{bmatrix}\cos\left(\frac{\pi}{3}\right) & -\sin\left(\frac{\pi}{3}\right)\\ \sin\left(\frac{\pi}{3}\right) & \cos\left(\frac{\pi}{3}\right)\end{bmatrix}$ 实现。

根据题意编写 MATLAB 代码如下：

```
function koch(a1,b1,a2,b2,n)
% koch(0,1,8,2,5)
% a1,b1,a2,b2 为初始线段两端点坐标,n 为迭代次数
a1 = 0;
b1 = 1;
a2 = 8;
b2 = 2;
n = 5;
[A,B] = sub_koch1(a1,b1,a2,b2);
for i = 1:n
    for j = 1:length(A)/5;
        w = sub_koch2(A(1 + 5 * (j - 1):5 * j),B(1 + 5 * (j - 1):5 * j));
        for k = 1:4
        [AA(5 * 4 * (j - 1) + 5 * (k - 1) + 1:5 * 4 * (j - 1) + 5 * (k - 1) + 5),...
            BB(5 * 4 * (j - 1) + 5 * (k - 1) + 1:5 * 4 * (j - 1) + 5 * (k - 1) + 5)]...
            = sub_koch1(w(k,1),w(k,2),w(k,3),w(k,4));
        end
   end
   A = AA;
   B = BB;
end
plot(A,B)
hold on
axis equal

function [A,B] = sub_koch1(ax,ay,bx,by)
% 由以(ax,ay),(bx,by)为端点的线段生成新的中间
% 三点坐标并把这五点横、纵坐标依次存储在数组中
cx = ax + (bx - ax)/3;
cy = ay + (by - ay)/3;
ex = bx - (bx - ax)/3;
```

```
ey = by - (by - ay)/3;
L = sqrt((ex - cx).^2 + (ey - cy).^2);
alpha = atan((ey - cy)./(ex - cx));
if (ex - cx)< 0
    alpha = alpha + pi;
end
dx = cx + cos(alpha + pi/3) * L;
dy = cy + sin(alpha + pi/3) * L;
A = [ax,cx,dx,ex,bx];
B = [ay,cy,dy,ey,by];

function w = sub_koch2(A,B)
a11 = A(1);
b11 = B(1);
a12 = A(2);
b12 = B(2);
a21 = A(2);
b21 = B(2);
a22 = A(3);
b22 = B(3);
a31 = A(3);
b31 = B(3);
a32 = A(4);
b32 = B(4);
a41 = A(4);
b41 = B(4);
a42 = A(5);
b42 = B(5);
w = [a11,b11,a12,b12;a21,b21,a22,b22;a31,b31,a32,b32;a41,b41,a42,b42];
```

运行代码后，得到 Koch 分形三角曲线如图 17-2 所示。

如果将一条直线形成如图 17-3 所示图形，编写 MATLAB 代码，实现 Koch 分形凸形曲线。

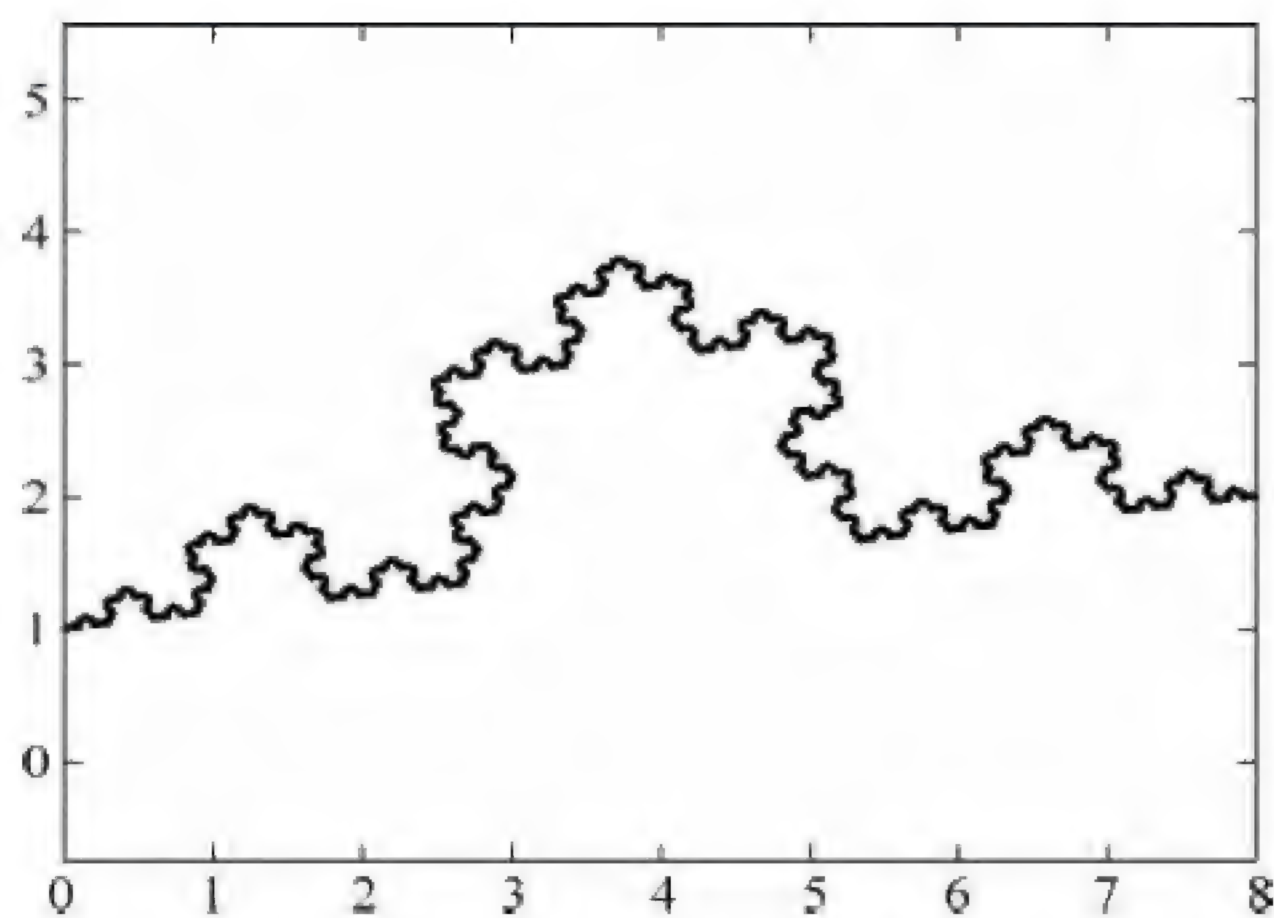


图 17-2　Koch 分形三角曲线



图 17-3　凸形线

此时，旋转矩阵为

$$A=\begin{vmatrix}\cos\left(\frac{\pi}{2}\right) & -\sin\left(\frac{\pi}{2}\right)\\ \sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right)\end{vmatrix}=\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}$$

编写 MATLAB 代码如下：

```
clear all
clc
%初始点坐标
p=[1 1;10 10];
n=2;                          %结点数
A=[0 -1;1 0];                 %旋转矩阵
for k=1:4
    d=diff(p)/3;
    m=5*n-4;                  %迭代公式
    q=p(1:n-1,:);             %以原点为起点，前 n-1 个点的坐标为终点形成向量
    p(6:5:m,:)=p(2:n,:);      %迭代后处于 5k+1 位置上的点的坐标为迭代前的相应坐标
    p(2:5:m,:)=q+d;           %用向量方法计算迭代后处于 5k+2 位置上的点的坐标
    p(3:5:m,:)=q+d+d*A';      %用向量方法计算迭代后处于 5k+3 位置上的点的坐标
    p(4:5:m,:)=q+2*d+d*A';    %用向量方法计算迭代后处于 5k+4 位置上的点的坐标
    p(5:5:m,:)=q+2*d;         %用向量方法计算迭代后处于 5k 位置上的点的坐标
    n=m;                      %迭代后新的结点数目
end
figure
plot(p(:,1),p(:,2))
axis([0 10 0 10])
```

运行代码后，得到 Koch 分形凸形曲线如图 17-4 所示。

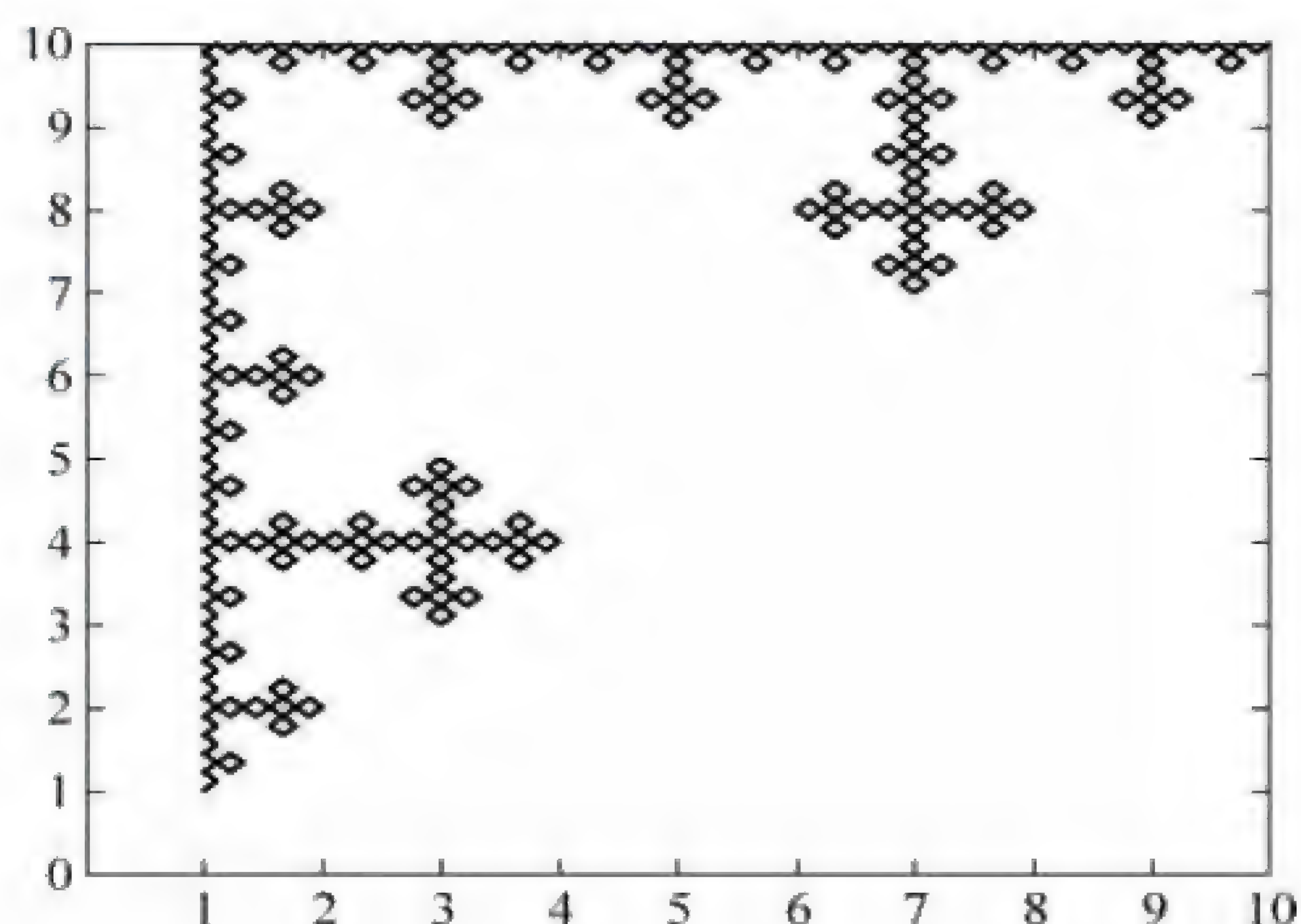


图 17-4　Koch 分形凸形曲线

17.2 二维分形维数的 MATLAB 应用

分形几何的主要工具就是其许多形式的维数。一般地，一条(光滑的)曲线称为一维，而一个曲面称为二维。

事实上，对于分形集来说，它们的维数是分数。分形可以分为规则分形和不规则分形。它们的维数计算方法也有很多，这里我们主要介绍不规则分形的盒维数计算方法。

假设有一个平面分形体为 A，盒子边长为 ε，$N_A(\varepsilon)$ 为至少包含一个 A 中点的盒子数。不断减小 ε，就可以得到一系列 $N_A(\varepsilon)$。如果

$$\lim_{\varepsilon \to 0} \frac{\ln N_A(\varepsilon)}{-\ln \varepsilon}$$

存在，则称这个极限值为集 A 的盒维数，记为 $\dim_B(A)$。

盒维数法也适用于一维和三维的不规则分形。对于一维空间中的，用等分的直线段来测量；对三维空间中的分形，可以用等分的小立方体网格进行测量。

【例 17-2】 利用 MATLAB 实现分形树图像。

解： 分形树是一条直线分形为图 17-5 所示的图像。

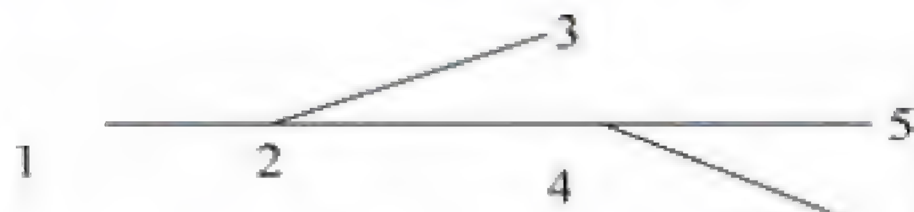


图 17-5　单个树形图

首先使用函数方法，实现分形树的 MATLAB 代码如下：

```
function tree(n,a,b)
n = 9;
a = pi/9;
b = pi/9;
x1 = 0;
y1 = 0;
x2 = 0;
y2 = 1;
plot([x1,x2],[y1,y2])
hold on
[X,Y] = tree1(x1,y1,x2,y2,a,b);
hold on
W = tree2(X,Y);
w1 = W(:,1:4);
w2 = W(:,5:8);
%w 为 2^k*4 维矩阵,存储第 k 次迭代产生的分支两端点的坐标
%w 的第 i(i=1,2,…,2^k)行数字对应第 i 个分支两端点的坐标
w = [w1;w2];
for k = 1:n
        for i = 1:2^k
        [X,Y] = tree1(w(i,1),w(i,2),w(i,3),w(i,4),a,b);
```

```
            W(i,:) = tree2(X,Y);
            end
      w1 = W(:,1:4);
      w2 = W(:,5:8);
      w = [w1;w2];
end

%把由函数 tree1 生成的 X,Y 顺次划分为两组,分别对应两分支两个端点的坐标,并存储在一维
数组中
function w = tree2(X,Y)
a1 = X(1);b1 = Y(1);
a2 = X(2);b2 = Y(2);
a3 = X(1);b3 = Y(1);
a4 = X(3);b4 = Y(3);
w = [a1,b1,a2,b2,a3,b3,a4,b4];
```

运行代码后,得到树形图如图 17-6 所示。

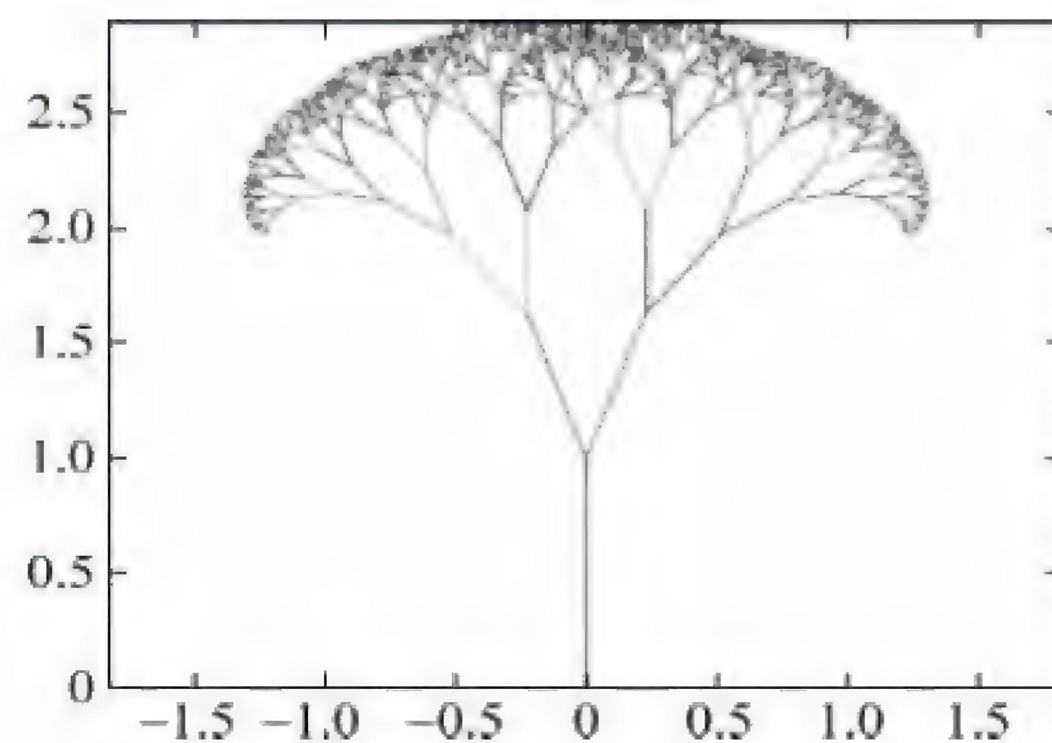


图 17-6　树形图

其次,还可以根据旋转矩阵编写如下代码:

```
clear all
clc
p = [1 1;9 9];
n = 2;                          %结点数
A = [cos(pi/3) - sin(pi/3);sin(pi/3) cos(pi/3)];
B = [cos( - pi/3) - sin( - pi/3);sin( - pi/3) cos( - pi/3)];
%旋转矩阵 A 对应于第一次逆时针旋转 60 度,旋转矩阵 B 对应于第二次顺时针旋转 60 度
for k = 1:5
    d = diff(p)/3;
    d1 = d(1:2:n,:);            %取每条线段对应的向量
    m = 5 * n;                  %迭代公式
    q1 = p(1:2:n - 1,:);
    p(10:10:m,:) = p(2:2:n,:);
    p(1:10:m,:) = p(1:2:n,:);      %迭代后处于 10k 与 10k + 1 位置上的点的坐标为迭代前的相
                                    应坐标
p(2:10:m,:) = q1 + d1;
%用向量方法计算迭代后处于 10k + 2,10k + 3,10k + 5 位置上的点的坐标,都相同
```

```
    p(3:10:m,:) = p(2:10:m,:);
    p(4:10:m,:) = q1 + d1 + d1 * A'; % 用向量方法计算迭代后处于 10k + 4 位置上的点的坐标
    p(5:10:m,:) = p(2:10:m,:);
p(6:10:m,:) = q1 + 2 * d1;
% 用向量方法计算迭代后处于 10k + 6,10k + 7,10k + 9 位置上的点的坐标,都相同
    p(7:10:m,:) = p(6:10:m,:);
    p(8:10:m,:) = q1 + 2 * d1 + d1 * B';
    p(9:10:m,:) = p(6:10:m,:);
    n = m;                          % 迭代后新的结点数目
end
plot(p(:,1),p(:,2))                 % 绘出每相邻两个点的连线
axis([0 10 0 10])
```

运行代码后，得到另一种树形图如图 17-7 所示。

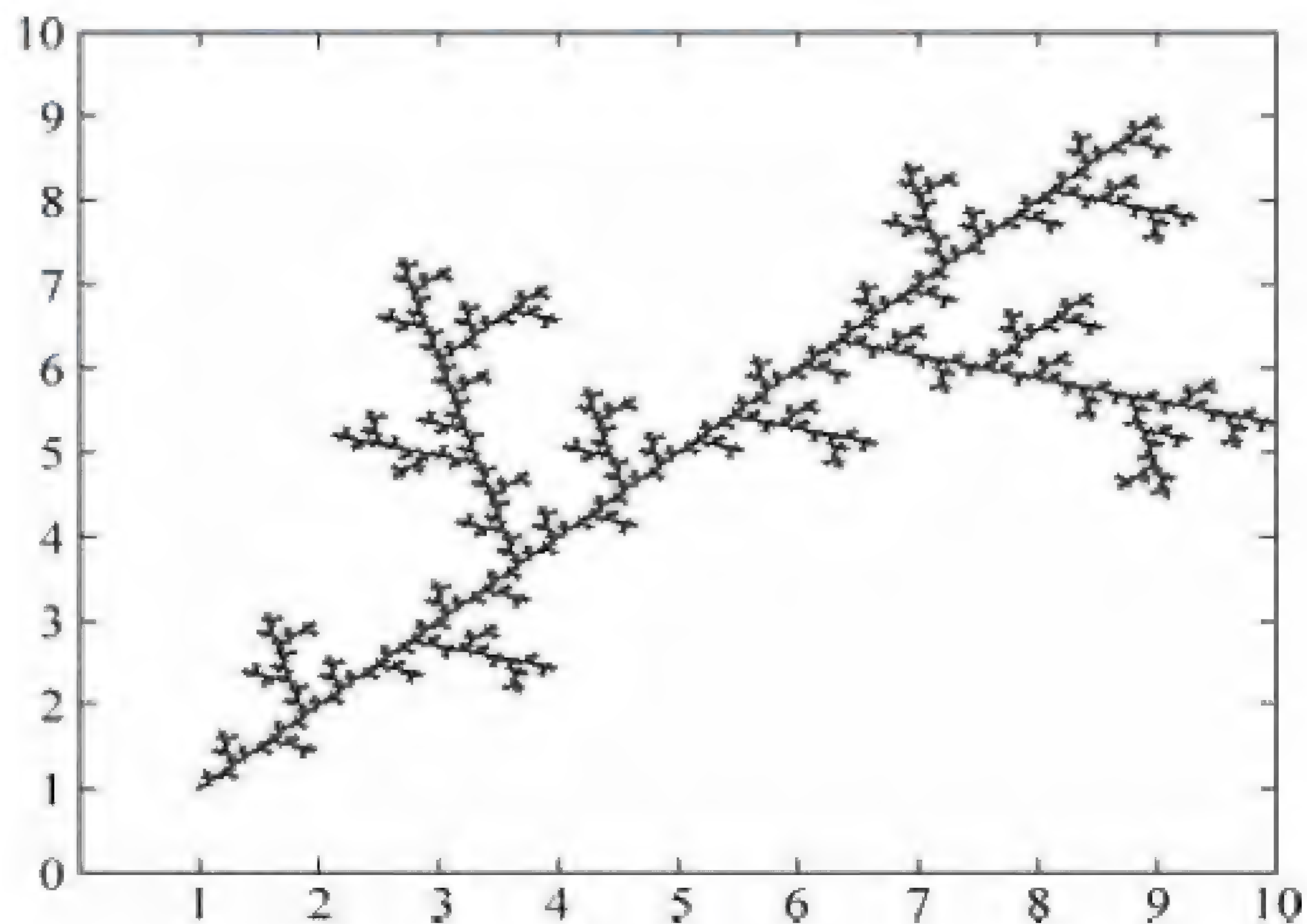


图 17-7　树形图

【例 17-3】 使用 MATLAB 编写 IFS 算法实现 Sierpinski 三角形程序、Julia 集程序和 Helix 曲线。

解：IFS 算法(iterator function system)是一种分形几何系统，主要通过仿射坐标转换来生成几何系统，仿射坐标转换是旋转、扭曲、平移三种效果的叠加。

1. Sierpinski 三角形的 MATLAB 实现程序如下：

```
function sierpinski_ifs(n,w1,w2,w3)
% w1,w2,w3 出现频率
n = 1000;
w1 = 1/4;
w2 = 1/4;
w3 = 1/4;
M1 = [0.5  0  0  0  0.5  0];
M2 = [0.5  0  0.5  0  0.5  0];
M3 = [0.5  0  0.25  0  0.5  0.5];
x = 1;
```

```
y = 1;
%r 为[0,1]区间内产生的 n 维随机数组
r = rand(1,n);
B = zeros(2,n);
k = 1;
% 当 0 < r(i)< 1/3 时,进行 M1 对应的压缩映射
% 当 1/3 = < r(i)< 2/3 时,进行 M2 对应的压缩映射
% 当 2/3 = < r(i)< 1 时,进行 M3 对应的压缩映射
for i = 1:n
    if r(i)< w1
        a = M1(1);
        b = M1(2);
        e = M1(3);
        c = M1(4);
        d = M1(5);
        f = M1(6);
    else if r(i)< w1 + w2
        a = M2(1);
        b = M2(2);
        e = M2(3);
        c = M2(4);
        d = M2(5);
        f = M2(6);
        else if r(i)< w1 + w2 + w3
        a = M3(1);
        b = M3(2);
        e = M3(3);
        c = M3(4);
        d = M3(5);
        f = M3(6);
            end
        end
    end
x = a * x + b * y + e;
y = c * x + d * y + f;
B(1,k) = x;
B(2,k) = y;
k = k + 1;
end
plot(B(1,:),B(2,:),'.','markersize',0.2)
```

运行代码后,得到 Sierpinski 三角形如图 17-8 所示。

2. Julia 集的 MATLAB 实现程序如下:

```
function julia_ifs(n,cx,cy)
n = 1000;
cx = - 0.88;
```

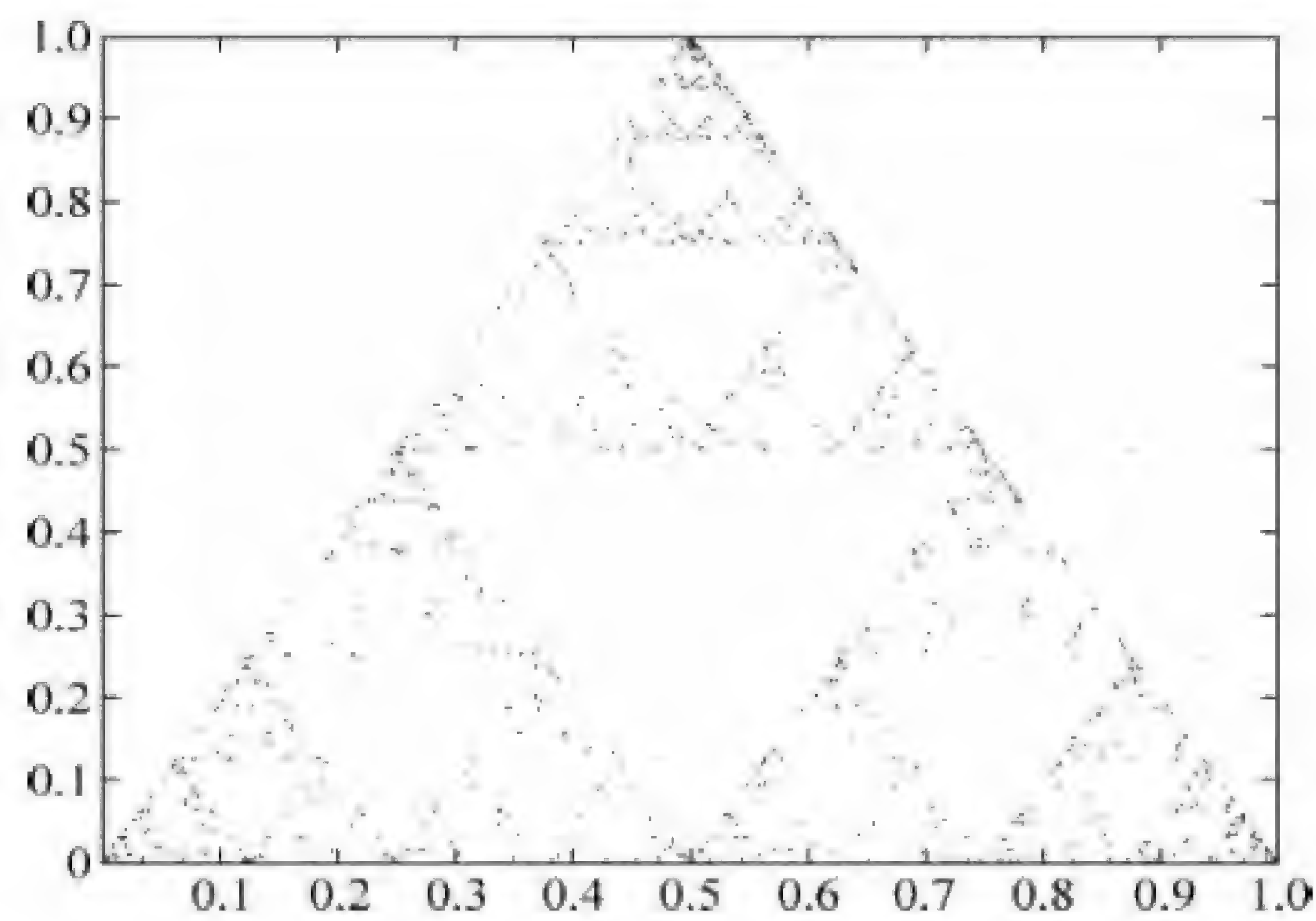


图 17-8　Sierpinski 三角形

```
cy = 0.09;
x = 0;
y = 0;
B = zeros(2,n);
k = 1;
%A 为产生的服从标准正态分布的 n 维随机数组
A = randn(1,n);
for i = 1:n
    wx = x - cx;
    wy = y - cy;
    if wx > 0
        alpha = atan(wy/wx);
    end
    if wx < 0
        alpha = pi + atan(wy/wx);
    end
    if wx == 0
        alpha = pi/2;
    end
    alpha = alpha/2;
    r = sqrt(wx^2 + wy^2);
      if A(i)< 0
        r = - sqrt(r);
      else
        r = sqrt(r);
      end
    x = r * cos(alpha);
    y = r * sin(alpha);
    B(1,k) = x;
    B(2,k) = y;
    k = k + 1;
end
plot(B(1,:),B(2,:),'.','markersize',0.1)
```

运行代码后，得到Julia集图形如图17-9所示。

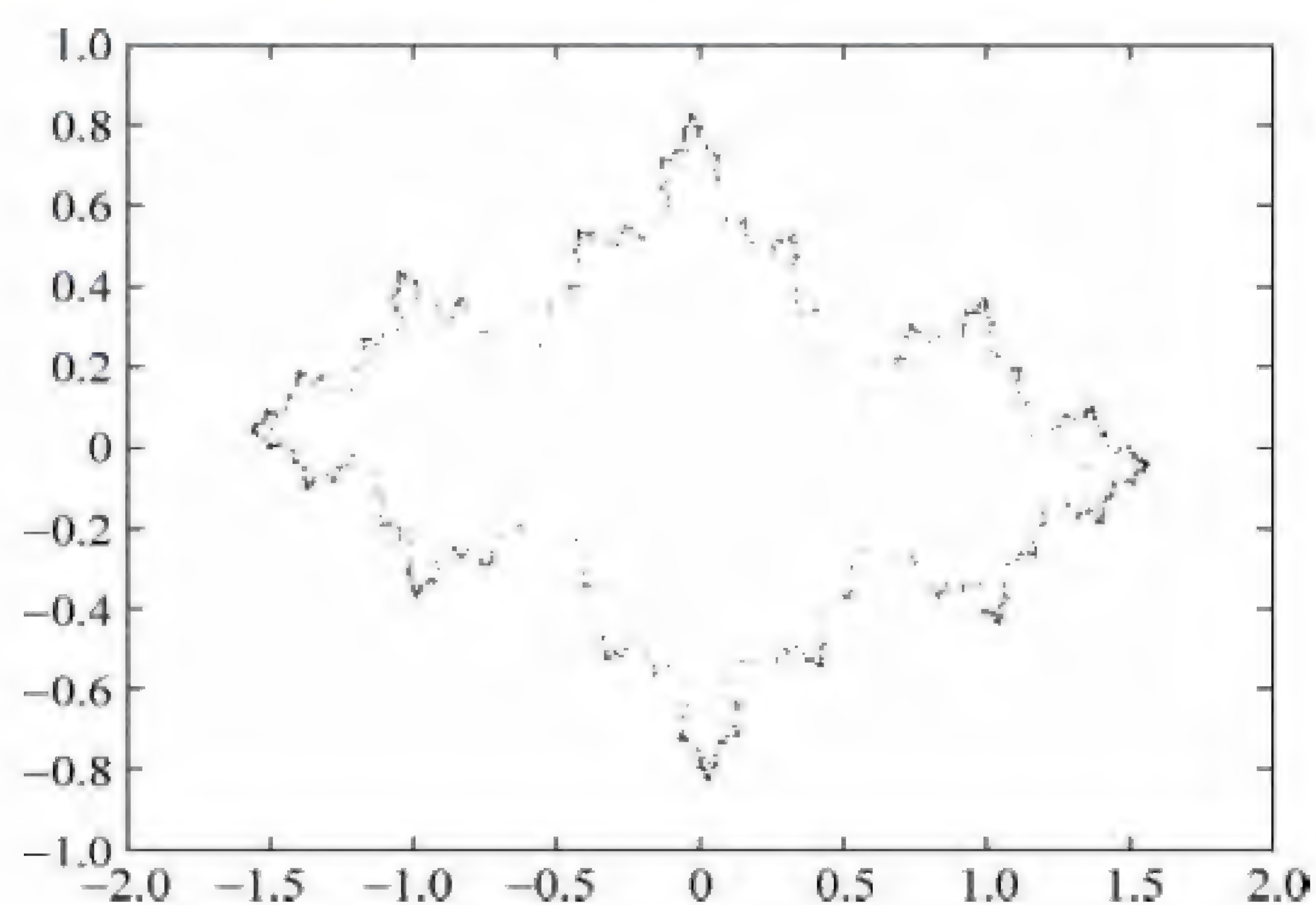


图 17-9　Julia 集图形

3. Helix 曲线的 MATLAB 实现代码如下：

```
function helix_ifs(n,w1,w2,w3)
%w1,w2,w3 为出现频率
n = 20000;
w1 = 0.7;
w2 = 0.08;
w3 = 0.08;
M1 = [0.786879  -0.426242  1.758647  0.242624  0.853868  1.408465];
M2 = [-0.121112  0.255276  -6.724254  0.053203  0.05283  1.377236];
M3 = [0.181418  -0.136324  6.082307  0.092309  0.181218  1.568025];
x = 0;
y = 0;
%r 为[0,1]区间内产生的 n 维随机数组
r = rand(1,n);
B = zeros(2,n);
k = 1;
%当 0<r(i)<1/3 时,进行 M1 对应的压缩映射
%当 1/3=<r(i)<2/3 时,进行 M2 对应的压缩映射
%当 2/3=<r(i)<1 时,进行 M3 对应的压缩映射
for i = 1:n
    if r(i)<w1
        a = M1(1);
        b = M1(2);
        e = M1(3);
        c = M1(4);
        d = M1(5);
        f = M1(6);
    else if r(i)<w1 + w2
        a = M2(1);
        b = M2(2);
```

```
            e = M2(3);
            c = M2(4);
            d = M2(5);
            f = M2(6);
            else if r(i)< w1 + w2 + w3
            a = M3(1);
            b = M3(2);
            e = M3(3);
            c = M3(4);
            d = M3(5);
            f = M3(6);
            end
        end
        end
x = a * x + b * y + e;
y = c * x + d * y + f;
B(1,k) = x;
  B(2,k) = y;
  k = k + 1;
end
plot(B(1,:),B(2,:),'.','markersize',0.1)
```

运行代码后，得到 Helix 曲线如图 17-10 所示。

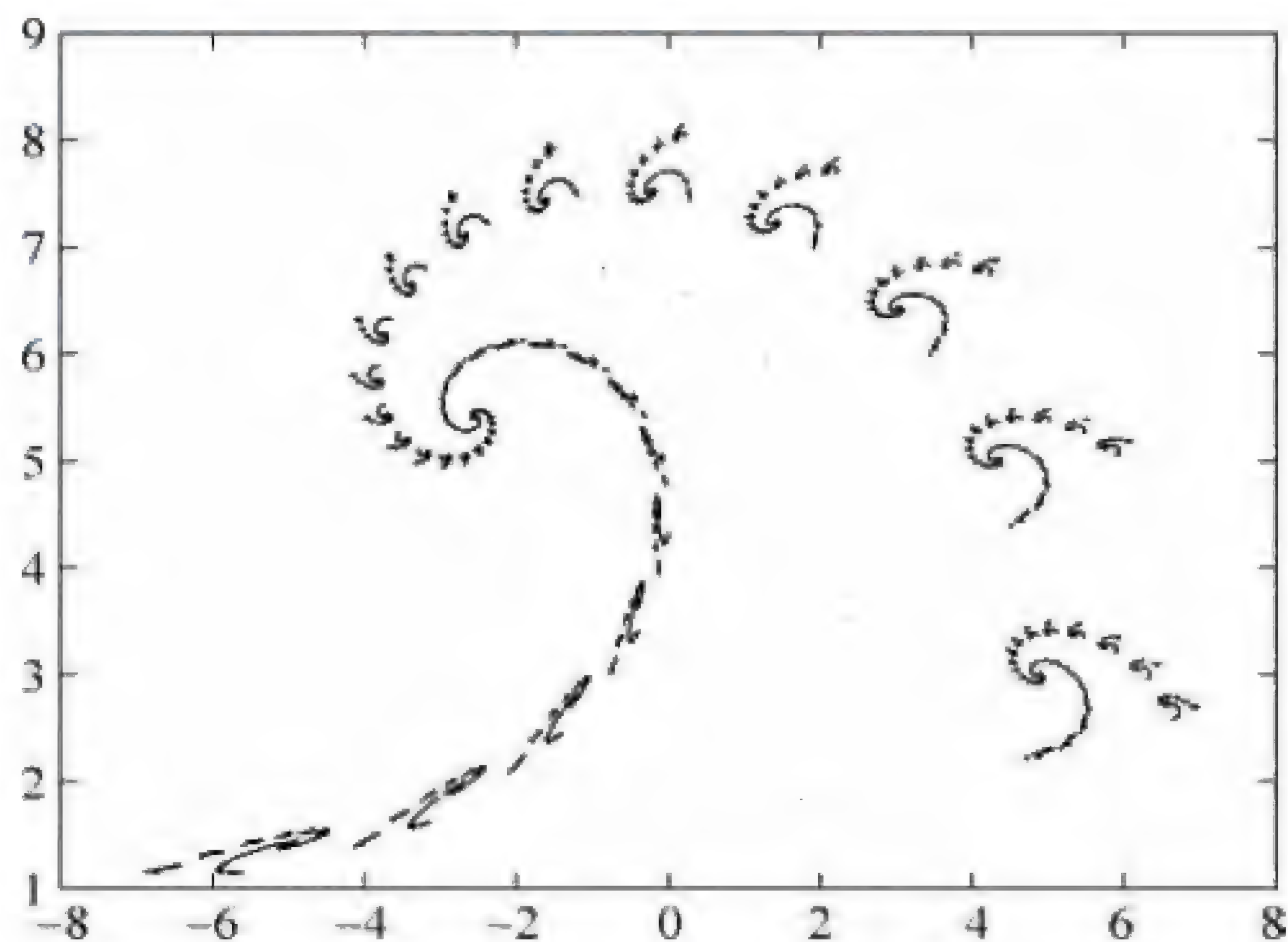


图 17-10　Helix 曲线

【例 17-4】 用元胞自动机算法画 Sierpinski 三角形。

解：元胞自动机分为一维元胞自动机和二维元胞自动机，下面分别用一维和二维元胞自动机求解 Sierpinski 三角形。

1. 一维元胞自动机

```
function sierpinski_ca1(m,n)
m = 1300;
```

```
n = 3500;
x = 0;
y = 0;
t = 1;
w = zeros(2,m * n);
s = zeros(m,n);
s(1,fix(n/3)) = 1;
for i = 1:m - 1
    for j = 2:n - 1
        if (s(i,j - 1) == 1&s(i,j) == 0&s(i,j + 1) == 0)|(s(i,j - 1) == 0&s(i,j) == 0&s(i,
j + 1) == 1)
            s(i + 1,j) = 1;
            w(1,t) = x + 3 + 3 * j;
            w(2,t) = y + 5 * i;
            t = t + 1;
        end
    end
end
plot(w(1,:),w(2,:),'.','markersize',1)
```

运行代码后，得到 Sierpinski 三角形如图 17-11 所示。

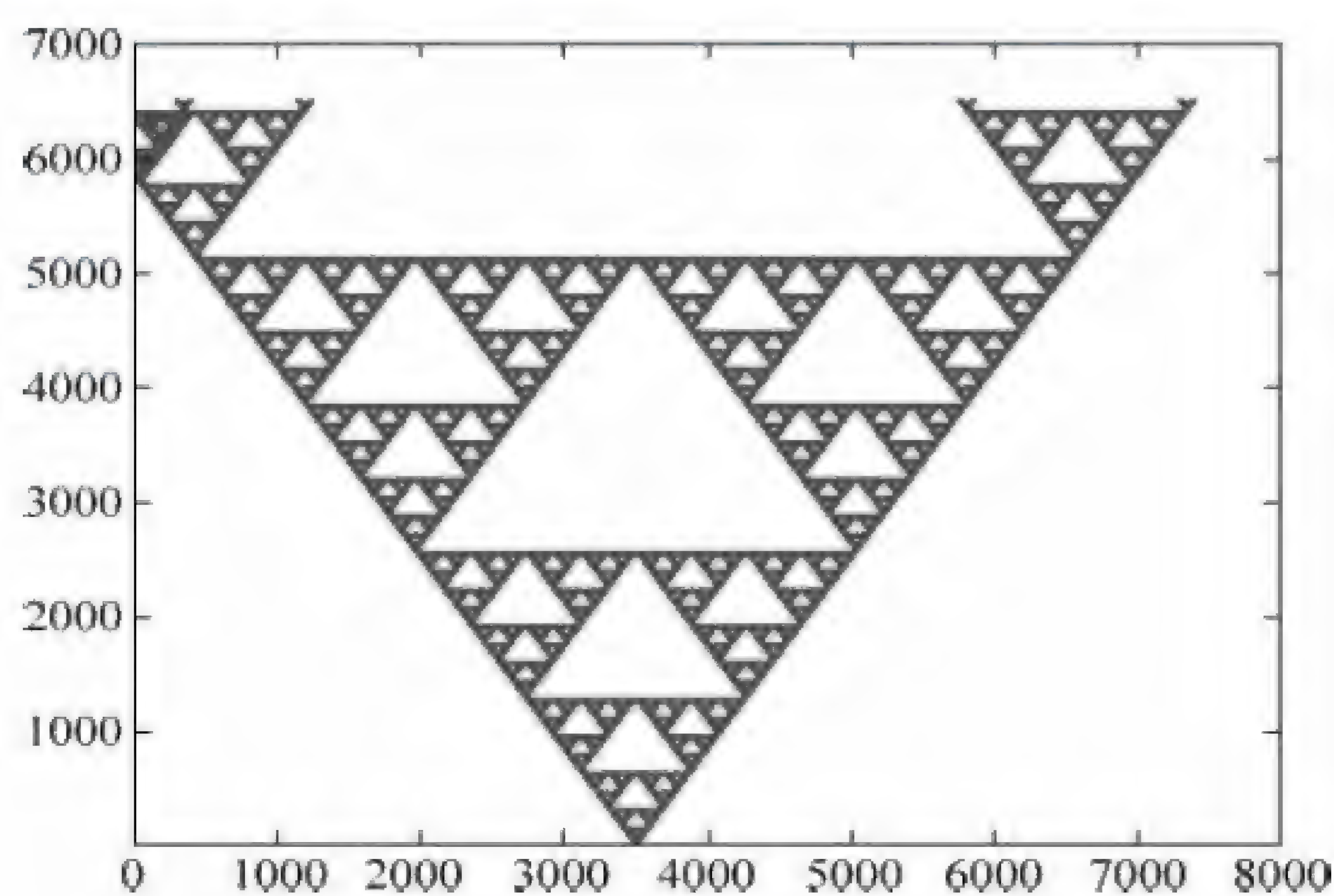


图 17-11　Sierpinski 三角形

2. 二维元胞自动机

```
function sierpinski_ca2(m,n)
m = 600;
n = 600;
t = 1;
w = zeros(2,m * n);
s = zeros(m,n);
s(m/2,n/2) = 1;
for i = [m/2: - 1:2,m/2:m - 1]
```

```
    for j = [n/2: - 1:2,n/2:n - 1]
        if mod(s(i - 1,j - 1) + s(i,j - 1) + s(i + 1,j - 1) + s(i - 1,j)...
                + s(i + 1,j) + s(i - 1,j + 1) + s(i,j + 1) + s(i + 1,j + 1),2) == 1
            s(i,j) = 1;
            w(1,t) = i;
            w(2,t) = j;
            t = t + 1;
        end
    end
end
plot(w(1,:),w(2,:),'.','markersize',0.1)
```

运行代码后，得到 Sierpinski 三角形如图 17-12 所示。

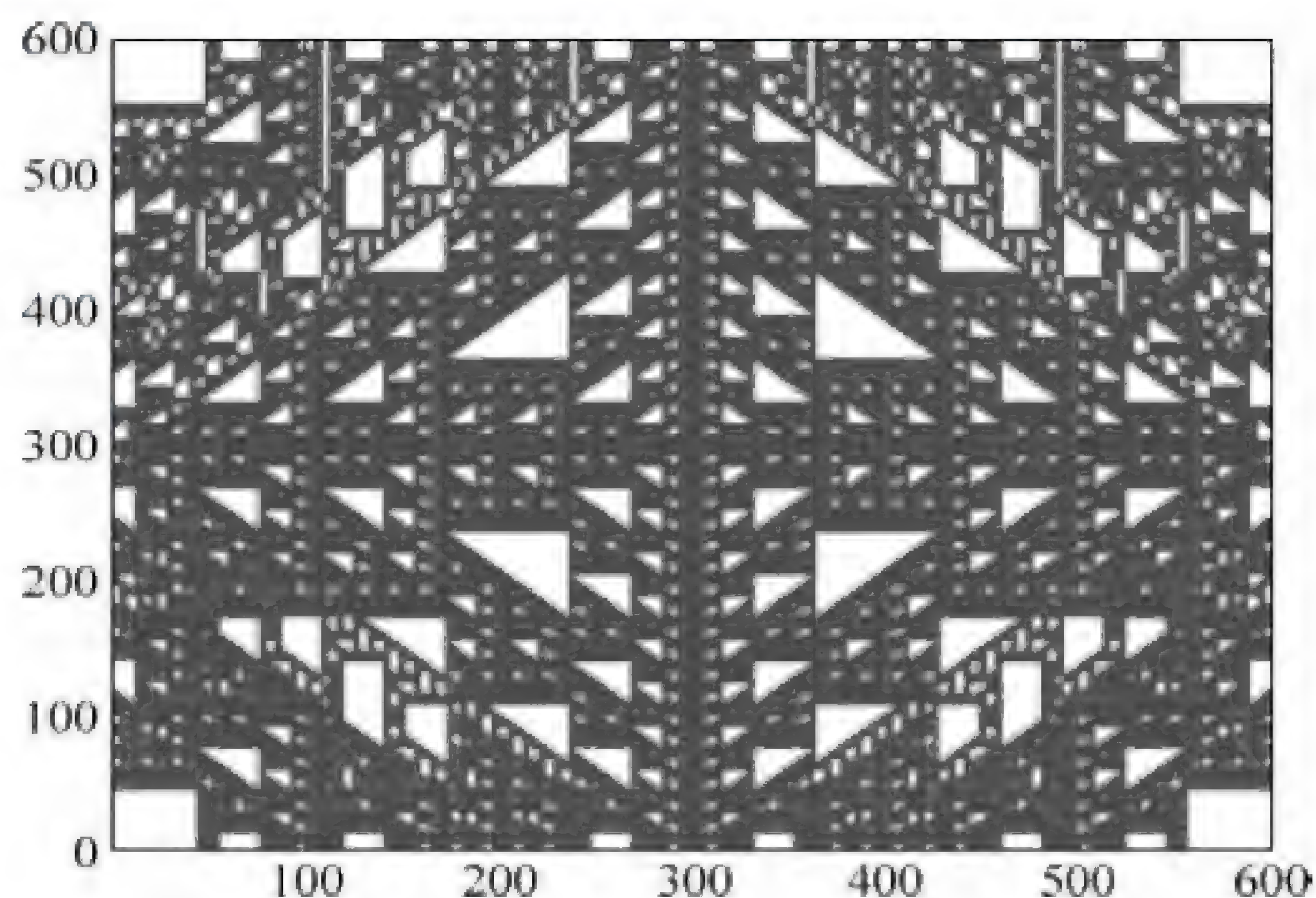


图 17-12　Sierpinski 三角形

17.3　分形插值算法的 MATLAB 应用

分形插值是根据分形几何自相似性原理和迭代函数系统 ifs 理论，将已知数据插值成具有自相似结构的曲线或曲面，其中每个局部都与整体自相似或统计自相似。因此，分形插值可以有效地避免传统插值方法对相邻插值点间局部变化特征的掩盖。

绝大多数情况下，图像处理是为了改善图像的视觉质量。因此，如何评价图像的质量是一个十分重要的问题。

在图像中，有些像素与相邻像素间灰度值存在突变，即存在灰度不连续性。这些具有灰度值突变的像素就是图像中描述对象的轮廓或纹理图像的边缘像素。在图像放大中，对这些具有不连续灰度特性的像素，如果采用常规的插值算法生成新增加的像素，势必会使放大图像的轮廓和纹理模糊，降低图像质量。

迭代系统函数对于处理规则分形图形有明显的优势，但是对于自然界现象用随机分形处理会更加形象逼真。

【例 17-5】 使用 MATLAB 首先对图 17-13 进行二值化处理，再分别用三种分形插值算法（最邻近插值、双线性插值、立方卷积插值）处理图形。



图 17-13　原始图像

解：根据题意要求，编写 MATLAB 代码如下：

```
clear all
clc
% 对图像二值化处理
I = imread('people.png');
I1 = rgb2gray(I);
level = graythresh(I1);
I2 = im2bw(I1,level);
I3 = ~I2;
I4 = bwareaopen(I3,50);
I5 = ~I4;
figure(1)
imshow(I5)
grid on
title('二值化图像','fontsize',12);

% 分别利用三种插值算法处理图像
I = imread('people.png');
figure(2)
subplot(2,2,1)
imshow(I);
grid on
title('原图','fontsize',12);

A = imresize(I,2,'nearest'); % 最邻近插值
subplot(2,2,2)
imshow(A);
grid on
title('最邻近插值图','fontsize',12);

B = imresize(I,2,'bilinear'); % 双线性插值
subplot(2,2,3)
imshow(B);
```

```
grid on
title('双线性插值图','fontsize',12);

C = imresize(I,2,'bicubic'); % 三次插值
subplot(2,2,4)
imshow(C);
grid on
title('三次插值图','fontsize',12);
```

运行代码后，得到二值化处理结果如图 17-14 所示，三种插值算法处理图像后的对比如图 17-15 所示。



图 17-14　二值化处理后的图像



图 17-15　三种插值算法处理图像前后对比图

【例 17-6】 已知如图 17-16 所示人脸和狗脸图片，使用分形插值算法调换图 17-16 中的人脸和狗脸。



图 17-16　人脸和狗脸图片

解：编写 MATLAB 代码如下：

```
clear all
clc
facedetection('people.png');

function rgbPic = bw2rgb(bwPic)

bwPicSize = size(bwPic);

rgbPic = zeros(bwPicSize(1),bwPicSize(2),3);

rgbPic(bwPic == 255) = 255;
rgbPic(:,:,2) = rgbPic(:,:,1);
rgbPic(:,:,3) = rgbPic(:,:,1);

rgbPic = im2uint8(rgbPic);

function facedetection(img_name)
I = imread('people.png');
I2 = imread('dog.png');

doge = imresize(I2,[size(I,1) size(I,2)]);
gray = rgb2gray(I);
YCbCr = rgb2ycbcr(I);
heigth = size(gray,1);
width = size(gray,2);
for i = 1:heigth
    for j = 1:width
        Y = YCbCr(i,j,1);
        Cb = YCbCr(i,j,2);
        Cr = YCbCr(i,j,3);
        if(Y < 80)
            gray(i,j) = 0;
        else
            if(skin(Y,Cb,Cr) == 1)
                gray(i,j) = 255;
            else
                gray(i,j) = 0;
            end
        end
    end
end
SE = strel('arbitrary',eye(5));
gray = imopen(gray,SE);
gray = imclose(gray,SE);

a1 = (255 - bw2rgb(gray))/255;
next1 = immultiply(I,a1);

a2 = bw2rgb(gray)/255;
```

```
next2 = immultiply(doge,a2);

add = imadd(next1,next2);
[L,num] = bwlabel(gray,8);
STATS = regionprops(L,'BoundingBox');
n = 1;
result = zeros(n,4);

figure,
subplot(1,2,1),
imshow(I),
title('原图 - 人脸')
subplot(1,2,2),
imshow(doge);
title('原图 - 狗脸')
figure,
subplot(1,2,1),
imshow(next1),
title('抠图效果')
subplot(1,2,2),
imshow(next2);
title('抠图效果')
figure,
imshow(add);
title('调换人脸效果图')

hold on;
for i = 1:num
    box = STATS(i).BoundingBox;
    x = box(1);
    y = box(2);
    w = box(3);
    h = box(4);
    ratio = h/w;
    ux = uint8(x);
    uy = uint8(y);
    if ux > 1
        ux = ux - 1;
    end
    if uy > 1
        uy = uy - 1;
    end
    if w < 20 || h < 20 || w * h < 400
        continue
    elseif ratio < 2.0 && ratio > 0.6 && findeye(gray,ux,uy,w,h) == 1
        result(n,:) = [ux uy w h];
        n = n + 1;
    end
end
```

```
if size(result,1) == 1 && result(1,1) > 0
    rectangle('Position',[result(1,1),result(1,2),result(1,3),result(1,4)],'EdgeColor',
'r');
else
    for m = 1:size(result,1)
        m1 = result(m,1);
        m2 = result(m,2);
        m3 = result(m,3);
        m4 = result(m,4);

        if m1 + m3 < width && m2 + m4 < heigth
            rectangle('Position',[m1,m2,m3,m4],'EdgeColor','r');
        end
    end
end

function eye = findeye(bImage,x,y,w,h)
part = zeros(h,w);
for i = y:(y+h)
    for j = x:(x+w)
        if bImage(i,j) == 0
            part(i-y+1,j-x+1) = 255;
        else
            part(i-y+1,j-x+1) = 0;
        end
    end
end
[L,num] = bwlabel(part,8);

if num < 2
    eye = 0;
else
    eye = 1;
end

function result = skin(Y,Cb,Cr)
a = 25.39;
b = 14.15;
ecx = 1.72;
ecy = 2.43;
sita = 2.64;
cx = 108.38;
cy = 153.32;
xishu = [cos(sita) sin(sita); -sin(sita) cos(sita)];
if(Y > 230)
    a = 1.1 * a;
    b = 1.1 * b;
end
Cb = double(Cb);
```

```
Cr = double(Cr);
t = [(Cb - cx);(Cr - cy)];
temp = xishu * t;
value = (temp(1) - ecx)^2/a^2 + (temp(2) - ecy)^2/b^2;
if value > 1
    result = 0;
else
    result = 1;
end
```

运行代码后，得到调换脸部的效果如图 17-17 所示。



图 17-17　调换脸部的效果

本章小结

近年来，分形理论不断发展，运用到多个学科，主要原因是分形可以将以前不能定量描述或者难以描述的复杂对象，通过一种较为便捷的定量方法表达。

本章首先介绍了分形维数的基本概念，然后重点介绍了二维分形维数和分形插值算法在 MATLAB 中的应用，并举例说明。

第18章 经济金融最优化应用

经济学对理解与指导中国经济的改革与发展、对帮助人们在日常工作与生活中进行理性决策都具有十分重要的作用。

经济与金融专业的学习可为众多的职业选择打下坚实的基础。本章重点介绍经济金融与MATLAB融合应用。

学习目标：

（1）了解期权定价基本概念；

（2）掌握收益风险和有效前沿的计算方法；

（3）熟练掌握投资组合绩效分析方法；

（4）掌握久期和凸度的计算方法。

18.1 期权定价分析

期权合约的权利类型，包括认购期权和认沽期权两种类型。认购期权，是指买方有权在特定日期按照特定价格买入约定数量合约标的的期权合约。认沽期权，是指买方有权在特定日期按照特定价格卖出约定数量合约标的的期权合约。行权价格，是指期权合约规定的，在期权买方行权时买入或卖出合约标的的交易价格。

斯克尔斯与他的同事已故数学家费雪·布莱克(Fischer Black)20世纪在70年代初合作研究出了一个期权定价的复杂公式(看涨和看跌)。默顿扩展了原模型的内涵，使之同样运用在许多其他形式的金融交易中。瑞士皇家科学协会赞誉他们在期权定价方面的研究成果是今后25年经济科学中的最杰出贡献。

在经济金融中，有BLACK-SCHOLES期权定价模型，其假设条件包括：

（1）金融资产收益率服从对数正态分布(股票价格走势遵循几何布朗运动)；

（2）在期权有效期内，无风险利率和金融资产收益变量是恒定的；

（3）市场无摩擦，即不存在税收和交易成本；

（4）该期权是欧式期权，即在期权到期前不可实施；

（5）金融资产在期权有效期内无红利及其他所得(该假设后被

放弃)。

Black-Scholes 定价公式如下:

$$c = SN(d_1) - Le^{-rT}N(d_2)$$

其中,$d_1=\dfrac{\ln(S/L)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$,$d_2=\dfrac{\ln(S/L)+(r-\sigma^2/2)T}{\sigma\sqrt{T}}=d_1-\sigma\sqrt{T}$,$C$ 是期权初始合理价格,L 是期权交割价格,S 是所交易金融资产现价,T 是期权有效期,r 是连续复利计无风险利率,σ^2 是年度化方差(波动率),$N()$是正态分布变量的累积概率分布函数。

【例 18-1】 假设欧式股票期权,六个月后到期,执行价格 95 元,现价为 152 元,无股利支付,股价年化波动率为 56%,无风险利率为 6%,计算期权价格,并画出期权价格与波动率关系图。

解:根据 Black-Scholes 定价模型,编写 MATLAB 代码如下:

```
clear all
clc
Price = 152;
Strike = 95;
Rate = 0.06;
Time = 6/12;
Volatility = 0.56;
[CallDelta, PutDelta] = blsprice(Price, Strike, Rate, Time, Volatility)

% 期权价格与波动率关系分析
Volatility = 0.08:0.01:0.5;
N = length(Volatility)
Call = zeros(1,N);
Put = zeros(1,N);
for i = 1:N
    [Call(i), Put(i)] = blsprice(Price, Strike, Rate, Time, Volatility(i));
end
plot(Call,'b--');
hold on
plot(Put,'b');
xlabel('Volatility')
ylabel('price')
legend('期权价格','波动率')
title('期权价格与波动率关系')
```

运行后,得到结果

```
CallDelta =
   62.0860
PutDelta =
    2.2783
N =
    43
```

即期权价格为 2.2783,期权价格与波动率关系图如图 18-1 所示。

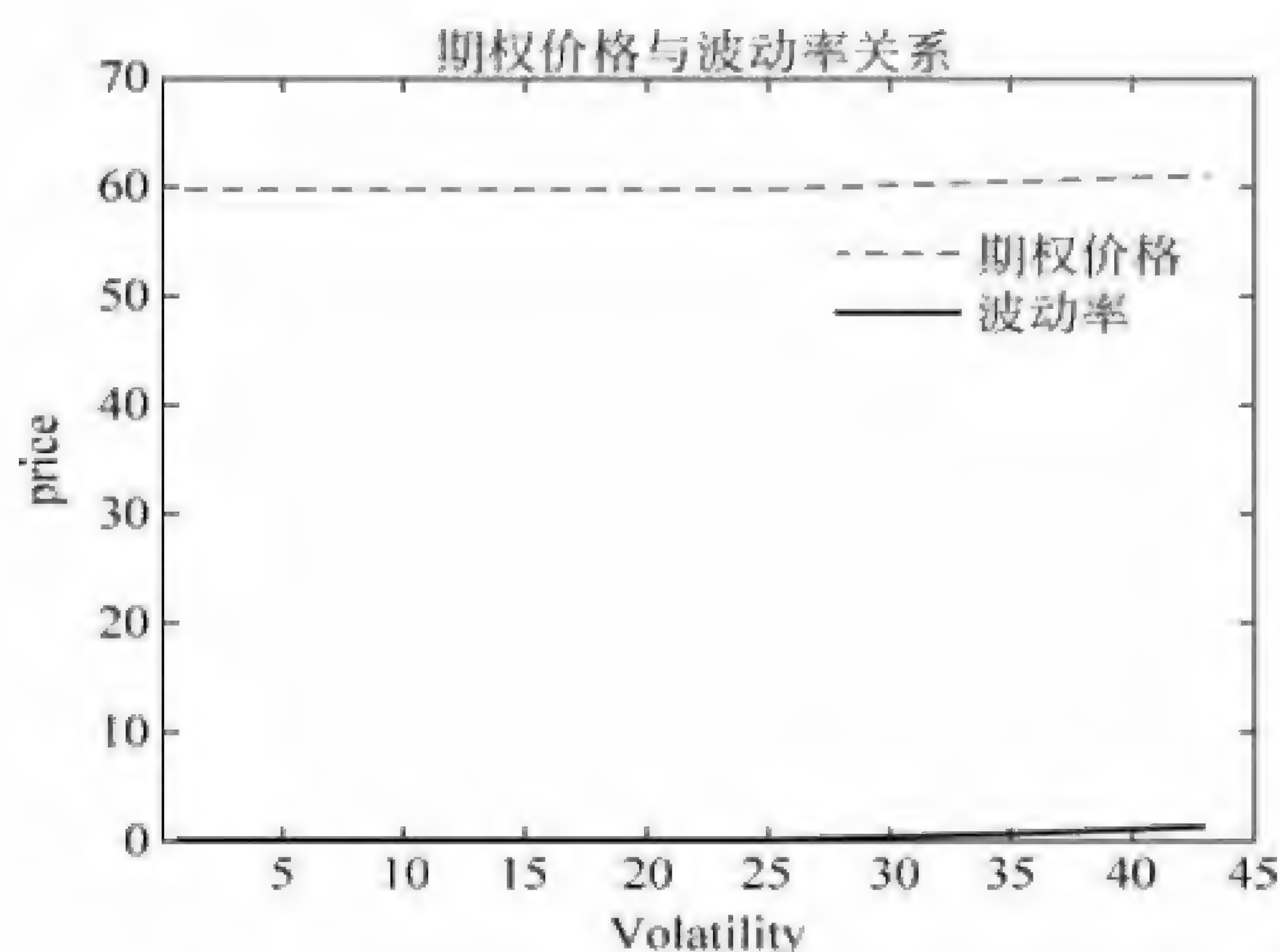


图 18-1　期权价格与波动率关系图

【例 18-2】 假设欧式股票期权，六个月后到期，执行价格 90 元，现价为 102 元，无股利支付，股价年化波动率为 55%，无风险利率为 8%，计算期权 Delta，并绘制 Price 与 Time、Detla 的三维关系。

解：编写 MATLAB 代码如下：

```
clear all
clc
Price = 102;
Strike = 90;
Rate = 0.08;
Time = 6/12;
Volatility = 0.55;
[CallDelta, PutDelta] = blsdelta(Price, Strike, Rate, Time, Volatility)

% 分析 Price、Time 和 Detla 三维关系图
Price = 60:1:102;
Strike = 90;
Rate = 0.08;
Time = (1:1:12)/12;
Volatility = 0.55;
[Price,Time] = meshgrid(Price,Time);
[Calldelta, Putdelta] = blsdelta(Price, Strike, Rate, Time, Volatility);
mesh(Price, Time, Putdelta);
xlabel('Stock Price ');
ylabel('Time (year)');
zlabel('Delta');
title('Price、Time 和 Detla 三维关系图')
```

运行后，得到结果为

```
CallDelta =
    0.7321
PutDelta =
   - 0.2679
```

Price、Time、Detla 的三维关系图如图 18-2 所示。

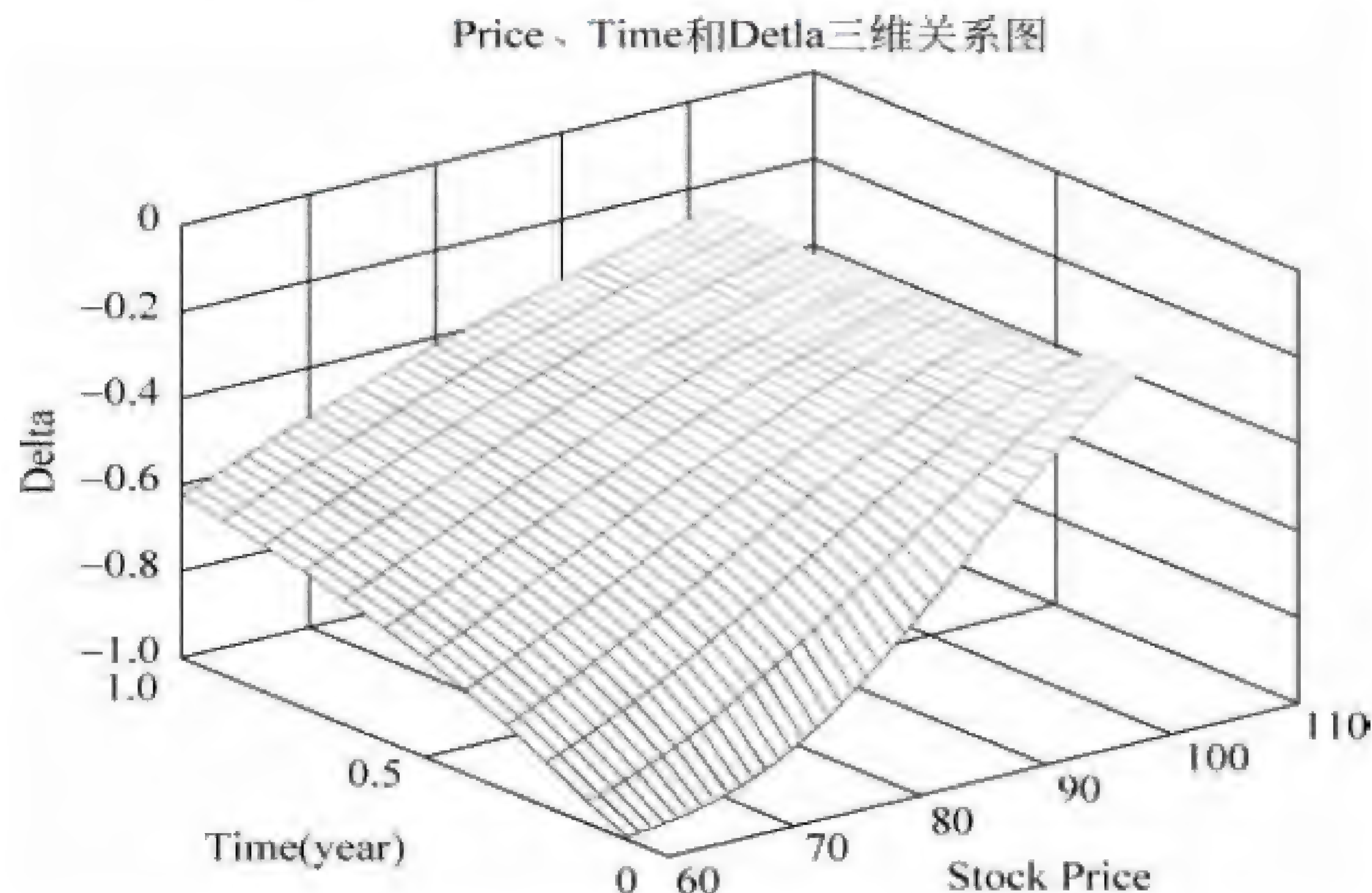


图 18-2 Price、Time、Detla 的三维关系图

期权的计算是从二叉树图的末端(时刻 T)开始向后倒退进行的。T 时刻期权的价值 V_n^N 已知。对于一个看涨期权来说,有

$$V_n^N = \max(S_n^N - K, 0)$$

对于一个看跌期权来说,有

$$V_n^N = \max(K - S_n^N, 0)$$

其中,$n=0,1,2,\cdots,N$, K 为执行价格。

在风险中性条件下,$T-\Delta t$ 时刻的每个结点上的期权值都可以用 T 时刻期权价值的期望值在时间 Δt 内用利率 r 贴现求出;同理,$T-2\Delta t$ 时刻的每个结点的期权值可以用 $T-\Delta t$ 时刻的期望值在 Δt 时间内用利率 r 贴现求出,其他结点依次类推。

如果对于美式期权,必须检查二叉树图的每个结点,以确定提前执行是否比继续持有 Δt 时间更有利。最后,向后倒推通过所有结点就求出了当前时刻的期权价值 V_0。

下面对美式期权定价问题进行研究。美式看涨期权被提前执行时,其内涵价值为

$$V_n^m = \max(S_n^m - K, 0) \quad n = 0,1,2,\cdots,m$$

对于看跌期权来说,有

$$V_n^m = \max(K - S_n^m, 0) \quad n = 0,1,2,\cdots,m$$

在 $m\Delta t$ 时刻从节点(m,n)向$(m+1)\Delta t$ 时刻的结点$(m+1,n+1)$移动的概率为 p;向$(m+1)\Delta t$ 时刻的结点$(m+1,n)$移动的概率为 $1-p$。

假设期权不提前执行,有

$$V_n^m = e^{-r\Delta t}[pV_{n+1}^{m+1} + (1-p)V_n^{m+1}]$$

若期权提前执行,必须与内涵价值相比较。那么对于看涨期权,有

$$V_n^m = \max\{\max(S_n^m - K,0), e^{-r\Delta t}[pV_{n+1}^{m+1} + (1-p)V_n^{m+1}]\}$$

对于看跌期权，有

$$V_n^m = \max\{\max(K - S_n^m,0)e^{-r\Delta t}[pV_{n+1}^{m+1} + (1-p)V_n^{m+1}]\}$$

【例 18-3】 假设欧式股票期权，六个月后到期，执行价格 115 元，现价为 100 元，无股利支付，股价年化波动率为 40%，无风险利率为 6%，红利率为 3%，计算美式期权价格矩阵和欧式期权价格矩阵。

解：编写 MATLAB 代码如下：

```
clear all
clc
S0 = 100;                          % 当前股价
K = 115;                           % 执行价格
r = 0.06;                          % 利率
T = 1;                             % 期权有效期
sigma = 0.4;                       % 波动率
q = 0.03;                          % 红利率
n = 1000;                          % 步数
dt = T/n;                          % 时间步长

%计算二叉树各参数
u = exp(sigma * sqrt(dt));         % 计算上升比率
d = 1/u;                           % 计算下降比率
p = (exp((r - q) * dt) - d)/(u - d);  % 计算上升的概率
%构造二叉树矩阵,i表示行数,j表示列数,Sx为股价矩阵,fx为期权的内在价值
for j = 1:n + 1
   for i = 1:j
      Sx(i,j) = S0 * (u^(j - i)) * (d^(i - 1));
      fx(i,j) = max(Sx(i,j) - K,0);
  end;
end;

%计算美式期权价格矩阵Afx和欧式期权价格矩阵Efx
for i = 1:n + 1                    % 到期时(j = n + 1)期权价格
   Afx(i,n + 1) = fx(i,n + 1);
   Efx(i,n + 1) = fx(i,n + 1);
end;

for jj = 1:n                       % 倒推前面各期(j = n - 1,n - 2, …,1)期权价格
   j = n + 1 - jj;
   for i = 1:j
      Efx(i,j) = exp( - r * dt) * (p * Efx(i,j + 1) + (1 - p) * Efx(i + 1,j + 1));
      Afx(i,j) = max(exp( - r * dt) * (p * Afx(i,j + 1) + (1 - p) * Afx(i + 1,j + 1)),fx(i,j));
   end;
end;

%输出结果
AmeOptionPrice = Afx(1,1)          % 美式期权价格矩阵
ErouOptionPrice = Efx(1,1)         % 欧式期权价格矩阵
```

运行后，得到结果为

```
AmeOptionPrice =
   10.8943

ErouOptionPrice =
   10.8943
```

【例 18-4】 假设欧式股票期权，三个月后到期，执行价格 80 元，现价为 105 元，无股利支付，股价年化波动率为 65%，无风险利率为 15%，编写 MATLAB 代码计算美式期权的二叉树定价矩阵。

解：编写 MATLAB 代码如下：

```
clear all
clc
Price = 105;
Strike = 80;
Rate = 0.15;
Time = 4/12;
flag = 1;
Increment = 1/12;
Volatility = 0.65;
[AssetPrice, OptionValue] = binprice(Price, Strike, Rate, Time, Increment, Volatility,
flag)
```

运行后，得到矩阵结果为

```
AssetPrice =
  105.0000  126.6718  152.8165  184.3575  222.4085
         0   87.0360  105.0000  126.6718  152.8165
         0         0   72.1453   87.0360  105.0000
         0         0         0   59.8023   72.1453
         0         0         0         0   49.5710

OptionValue =
   32.8817  50.6403  74.7917  105.3513  142.4085
         0  16.8591  28.9948   47.6655   72.8165
         0        0   5.7721   12.0126   25.0000
         0        0        0         0         0
         0        0        0         0         0
```

18.2 收益、风险和有效前沿的计算

在公司经营活动中，实际上所有的财务活动决策都有一个共同点，即需要估计预期的结果和影响这一结果不能实现的可能性。一般来说，预期的结果就是所谓的预期收益，而影响这一结果不能实现的可能性就是风险。

所谓收益(return)是指投资机会未来收入流量超过支出流量的部分,所谓风险(risk)是指预期收益发生变动的可能性,或者说是预期收益的不确定性。

有效前沿是一个经济术语,指对于一个理性的投资者而言,他们都是厌恶风险而偏好收益。对于相同的风险水平,他们会选择能提供最大收益率的组合;对于相同的预期收益率,他们会选择风险最小的组合;能同时满足这两个条件的投资组合就是有效集。

【例 18-5】 从 Wind 咨询金融终端分别下载三只股票(美好集团、石油化服和首开股份)从 2013 年年初至今的日收盘价,经过相关处理得出三只股票的收益率均值、标准差以及协方差矩阵等数据,如表 18-1 所示。现根据表格数据进行关于收益、风险和有效前沿计算。

表 18-1 三只股票数据

	收益率均值	收益率标准差	协方差矩阵		
美好集团 A	0.0017	0.0312	0.0010	0.0004	0.0005
石油化服 B	0.0015	0.0411	0.0004	0.0016	0.0003
首开股份 C	0.0005	0.0361	0.0005	0.0003	0.0013

(1) 假设等权重配置 A、B、C 三只股票,计算资产组合的风险和收益。

解:在 MATLAB 中编写以下代码:

```
clear all
clc
ExpReturn = [0.0017,0.0015,0.0005];
ExpCovariance = [0.0010,0.0004,0.0005;...
    0.0004,0.0016,0.0003;...
    0.0005,0.0003,0.0013];
PortWts = 1/3 * ones(1,3);
[PortRisk, PortReturn] = portstats(ExpReturn, ExpCovariance,PortWts)
```

运行后,得到结果为

```
PortRisk =
    0.0265

PortReturn =
    0.0012
```

(2) 当希望资产组合为最优组合,计算三只股票的最优配置。

解:依题意,编写以下 MATLAB 代码:

```
clear all
clc
ExpReturn = [0.0017,0.0015,0.0005];
ExpCovariance = [0.0010,0.0004,0.0005;...
    0.0004,0.0016,0.0003;...
```

```
    0.0005,0.0003,0.0013];
NumPorts = 10;
TargetReturn = [ 0.07; 0.08; 0.09 ];

p = Portfolio;
p = setAssetMoments(p, ExpReturn, ExpCovariance);
p = setDefaultConstraints(p);

PortWts = estimateFrontierByReturn(p, TargetReturn);
[PortRisk, PortReturn] = estimatePortMoments(p, PortWts);

disp('Efficient  Target');
disp([PortReturn, TargetReturn]);
disp('风险');
PortRisk % 风险
disp('收益');
PortReturn % 收益
disp('有效前沿矩阵');
PortWts % 有效前沿
```

运行后，得到结果为

```
Efficient  Target
    0.0017  0.0700
    0.0017  0.0800
    0.0017  0.0900

风险
PortRisk =
    0.0316
    0.0316
    0.0316

收益
PortReturn =
    0.0017
    0.0017
    0.0017

有效前沿函数
PortWts =
    1.0000  1.0000  1.0000
    0.0000  0.0000  0.0000
         0       0       0
```

即三只股票等额配置最优。

（3）当各个资产投资上限为50%，计算其有效前沿。

解：编写MATLAB代码如下：

```
clear all
clc
ExpReturn = [0.0017,0.0015,0.0005];
ExpCovariance = [0.0010,0.0004,0.0005;...
    0.0004,0.0016,0.0003;...
    0.0005,0.0003,0.0013];
NumPorts = 10;
AssetBounds = [0,0,0;0.5,0.5,0.5];
%[PortRisk, PortReturn, PortWts] = frontcon(ExpReturn, ExpCovariance, NumPorts, [ ],
AssetBounds)
LowerBound = AssetBounds(1,:);
UpperBound = AssetBounds(2,:);

p = Portfolio;
p = setAssetMoments(p, ExpReturn, ExpCovariance);
p = setDefaultConstraints(p);
p = setBounds(p, LowerBound, UpperBound);

PortWts = estimateFrontier(p, NumPorts);
[PortRisk, PortReturn] = estimatePortMoments(p, PortWts);

disp('风险');
PortRisk % 风险
disp('收益');
PortReturn % 收益
disp('有效前沿矩阵');
PortWts % 有效前沿
```

运行后,得到结果为

```
风险
PortRisk =
    0.0262
    0.0263
    0.0263
    0.0264
    0.0266
    0.0270
    0.0274
    0.0279
    0.0285
    0.0292

收益
PortReturn =
    0.0013
    0.0013
    0.0014
    0.0014
```

```
    0.0014
    0.0015
    0.0015
    0.0015
    0.0016
    0.0016

有效前沿矩阵
PortWts =
    0.4312  0.4579  0.4845  0.5000  0.5000  0.5000  0.5000  0.5000  0.5000  0.5000
    0.2661  0.2692  0.2724  0.2890  0.3242  0.3593  0.3945  0.4297  0.4648  0.5000
    0.3028  0.2729  0.2431  0.2110  0.1758  0.1407  0.1055  0.0703  0.0352       0
```

(4) 假如配置 A、B、C 三个资产，A 最大配置 60%，B 最大配置 70%，C 最大配置 50%，A 为资产集合一，B、C 组成资产集合二，资产集合一的最大配置为 70%，资产集合二的最大配置为 50%，资产集合一的配置不能超过资产集合二的 3 倍，编写 MATLAB 求解三个资产的最优配置。

解：编写 MATLAB 代码如下：

```
clear all
clc
ExpReturn = [0.0017,0.0015,0.0005];
ExpCovariance = [0.0010,0.0004,0.0005;...
    0.0004,0.0016,0.0003;...
    0.0005,0.0003,0.0013];
NumPorts = 5;
NumAssets = 3;
PVal = 1;
AssetMin = 0;
AssetMax = [0.6,0.7,0.5];
GroupA = [1 0 0];
GroupB = [0 1 1];
GroupMax = [0.7,0.5];
AtoBmax = 3;
ConSet = portcons('PortValue', PVal, NumAssets,'AssetLims',...
AssetMin, AssetMax, NumAssets, 'GroupComparison',GroupA, NaN,...
AtoBmax, GroupB,GroupMax );

A = ConSet(:,1:end-1);
b = ConSet(:,end);

p = Portfolio;
p = setAssetMoments(p, ExpReturn, ExpCovariance);
p = setInequality(p, A, b);

PortWts = estimateFrontier(p, NumPorts);
[PortRisk, PortReturn] = estimatePortMoments(p, PortWts);
```

```
disp('风险');
PortRisk % 风险
disp('收益');
PortReturn % 收益
disp('有效前沿矩阵');
PortWts % 有效前沿
```

运行后,得到结果为

```
风险

PortRisk =

    0.0262
    0.0264
    0.0267
    0.0273
    0.0284

收益

PortReturn =

    0.0013
    0.0014
    0.0015
    0.0015
    0.0016

有效前沿矩阵

PortWts =

    0.4312  0.4950  0.5588  0.6000  0.6000
    0.2661  0.2736  0.2812  0.3159  0.4000
    0.3028  0.2314  0.1600  0.0841  0.0000
```

18.3 投资组合绩效分析

自格雷厄姆和多德最早提出价值投资理论以来,在资本市场尤其是机构投资者中得到了广泛应用,并涌现出以巴菲特为代表的价值投资大师群体。传统理论认为,价值投资者首先通过评估某一金融资产的基础内在价值,并将之与市场价格进行比较,如果价格低于价值形成的安全边际,则进行买入持有操作。

在投资过程中,通过多个维度的基本指标分析寻找有估值优势,同时又有较强的持续稳定增长的股票进行投资。

一方面利用可横向比较的估值型因子指标筛选低估值股票,构建投资组合在市场波

动时的安全边界；另一方面，利用股票自身特有的成长指标筛选高成长股票，分享成长股潜力释放过程中的高收益机会。

【例 18-6】 已知三只股票(股 A、股 B、股 C)和一只指数(指 A)的部分日收盘价数据，根据这些数据对股 A、股 B、股 C 和指 A 进行投资组合绩效分析。

解：首先画出三只股票和一只指数曲线图，编写 MATLAB 代码如下：

```
clear all
clc
load GZ.mat
figure;
hold on
plot(GZ(:,1)/GZ(1,1),'g')
plot(GZ(:,2)/GZ(1,2),'b-.')
plot(GZ(:,3)/GZ(1,3),'r--')
plot(GZ(:,4)/GZ(1,4),'ko')
title('股票和指数曲线')
xlabel('时间')
ylabel('价格')
legend('指 A','股 A','股 B','股 C')
```

运行后，得到股票和指数曲线图如图 18-3 所示。

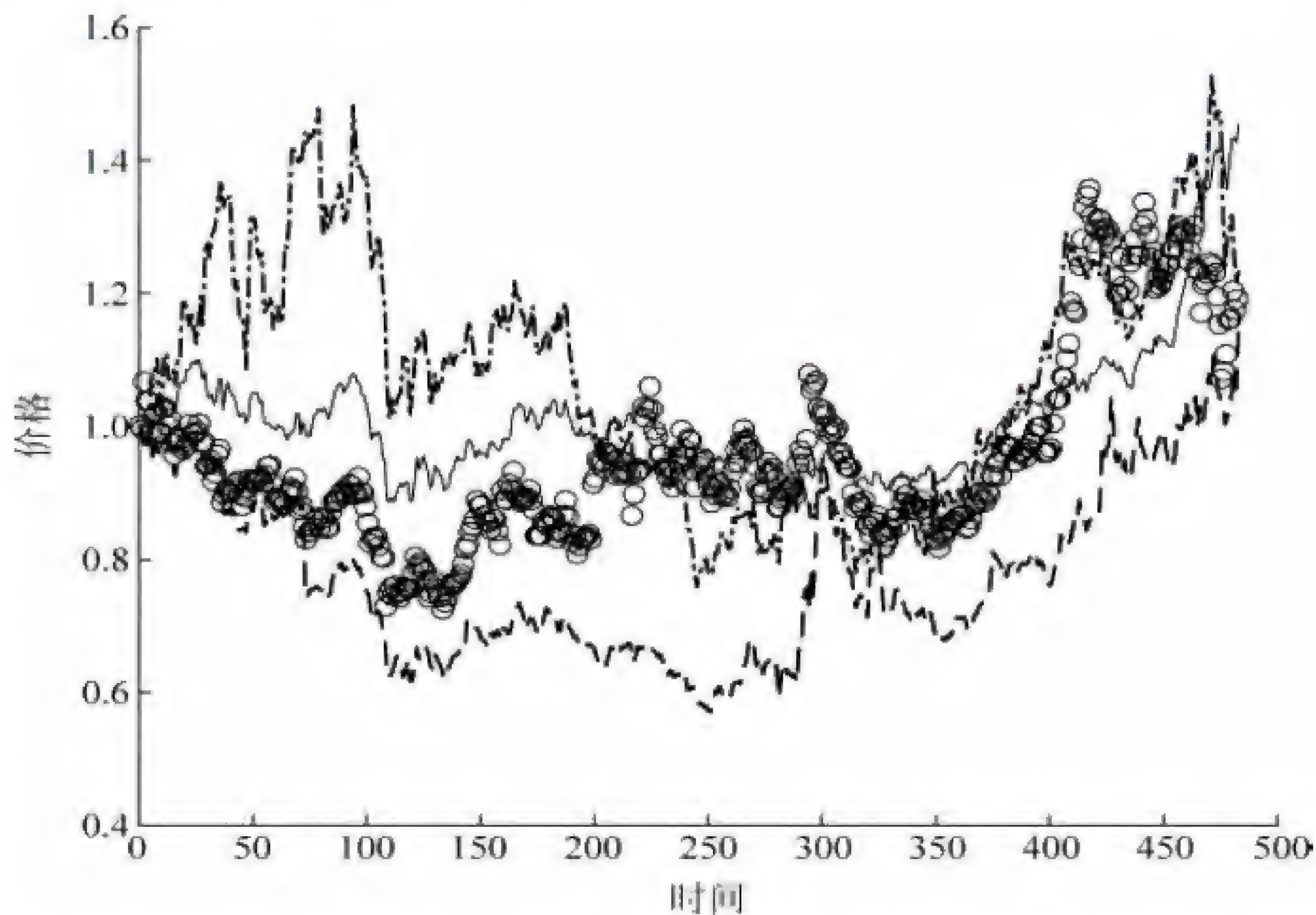


图 18-3 股票和指数曲线图

以指 A 作为市场收益，编写如下代码计算三家股票的 Alpha 值：

```
Rate = price2ret(GZ);
zz800 = Rate(:,1);
hljz = Rate(:,2);
hydc = Rate(:,3);
```

```
hfcj = Rate(:,4);
daynum = fix(length(Rate)/2);
cash = (1 + 0.03)^(1/daynum) - 1;
cash = cash * ones(daynum,1);
RatioHL2013 = daynum * portalpha(hljz(1:daynum),zz800(1:daynum),cash,'capm')
RatioHL2014 = daynum * portalpha(hljz(daynum + 1:2 * daynum),zz800(daynum + 1:2 * daynum),
cash,'capm')
RatioHY2013 = daynum * portalpha(hydc(1:daynum),zz800(1:daynum),cash,'capm')
RatioHY2014 = daynum * portalpha(hydc(daynum + 1:2 * daynum),zz800(daynum + 1:2 * daynum),
cash,'capm')
RatioHF2014 = daynum * portalpha(hfcj(1:daynum),zz800(1:daynum),cash,'capm')
RatioHF2013 = daynum * portalpha(hfcj(daynum + 1:2 * daynum),zz800(daynum + 1:2 * daynum),
cash,'capm')
```

运行后，得到结果为

```
RatioGA2013 =
    - 0.0969
RatioGA2014 =
    - 0.0413
RatioGB2013 =
    - 0.4380
RatioGB2014 =
     0.1599
RatioGC2014 =
     0.0314
RatioGC2013 =
    - 0.1979
```

计算股A、股B和股C的夏普比率。

```
Rate = price2ret(GZ);
hljz = Rate(:,2);
hydc = Rate(:,3);
hfcj = Rate(:,4);
daynum = fix(length(Rate)/2);
Cash = (1 + 0.03)^(1/daynum) - 1;
RatioGA2013 = sharpe(hljz(1:daynum),Cash)
RatioGA2014 = sharpe(hljz(daynum + 1:2 * daynum),Cash)
RatioGBDC2013 = sharpe(hydc(1:daynum),Cash)
RatioGBDC2014 = sharpe(hydc(daynum + 1:2 * daynum),Cash)
RatioGC2013 = sharpe(hfcj(1:daynum),Cash)
RatioGC2014 = sharpe(hfcj(daynum + 1:2 * daynum),Cash)
```

运行后得到股A、股B和股C的夏普比率分别为

```
RatioGA2013 =
    - 0.0272
RatioGA2014 =
```

```
    0.0599
RatioGBDC2013 =
   -0.1055
RatioGBDC2014 =
    0.0925
RatioGC2013 =
   -0.0089
RatioGC2014 =
    0.0306
```

以指A指数作为业绩比较基准，计算股A、股B和股C的信息比率。

```
Rate = price2ret(GZ);
zz800 = Rate(:,1);
hljz = Rate(:,2);
hydc = Rate(:,3);
hfcj = Rate(:,4);
daynum = fix(length(Rate)/2);
RatioGA2013 = inforatio(hljz(1:daynum),zz800(1:daynum))
RatioGA2014 = inforatio(hljz(daynum + 1:2 * daynum),zz800(daynum + 1:2 * daynum))
RatioGBDC2013 = inforatio(hydc(1:daynum),zz800(1:daynum))
RatioGBDC2014 = inforatio(hydc(daynum + 1:2 * daynum),zz800(daynum + 1:2 * daynum))
RatioGC2013 = inforatio(hfcj(1:daynum),zz800(1:daynum))
RatioGC2014 = inforatio(hfcj(daynum + 1:2 * daynum),zz800(daynum + 1:2 * daynum))
```

运行后得到股A、股B和股C的信息比率为

```
RatioGA2013 =
   -0.0181
RatioGA2014 =
   -0.0100
RatioGBDC2013 =
   -0.1238
RatioGBDC2014 =
    0.0362
RatioGC2013 =
    0.0079
RatioGC2014 =
   -0.0500
```

根据股A的数据计算最大回撤。编写MATLAB代码如下：

```
TRate = GZ(:,2)/GZ(1,2) - 1;
[MaxDD,MaxDDIndex] = maxdrawdown(TRate,'arithmetic')
plot(TRate)
hold on
plot(MaxDDIndex,TRate(MaxDDIndex),'r-o','MarkerSize',10)
title('根据股A的数据得到的最大回撤曲线')
```

运行后，得到计算结果为

```
MaxDD =
    0.7223
MaxDDIndex =
    94
   245
MaxDD =
    2.3110
MaxDDIndex =
   359
   417
```

最大回撤曲线如图 18-4 所示。

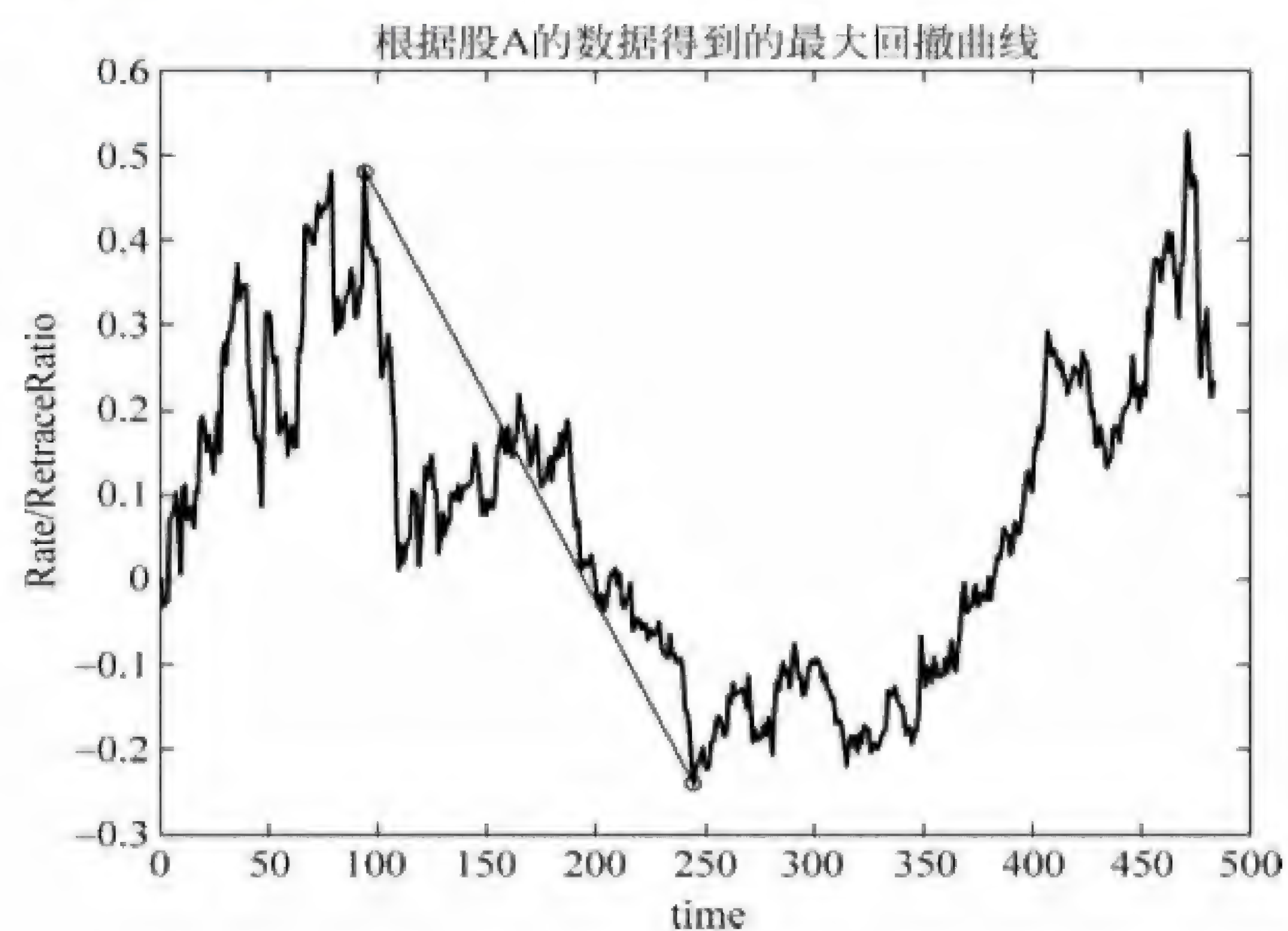


图 18-4　最大回撤曲线

根据指 A 数据画出最大收益回撤，编写代码如下：

```
ZZ800price = GZ(:,1);
N = length(ZZ800price);
RetraceRatio = zeros(N,1);
for i = 2:N
    C = max(ZZ800price(1:i));
    if C == ZZ800price(i)
         RetraceRatio(i) = 0;
    else
        RetraceRatio(i) = (ZZ800price(i) - C)/C;
    end
end
TRate = ZZ800price/ZZ800price(1) - 1;
f = figure;
fill([1:N,N],[RetraceRatio;0],'r')
```

```
hold on
plot(TRate);
xlabel('time')
ylabel('Rate/RetraceRatio')
title('根据指 A 数据画出最大收益回撤图')
```

运行后,得到根据指 A 数据画出最大收益回撤图如图 18-5 所示。

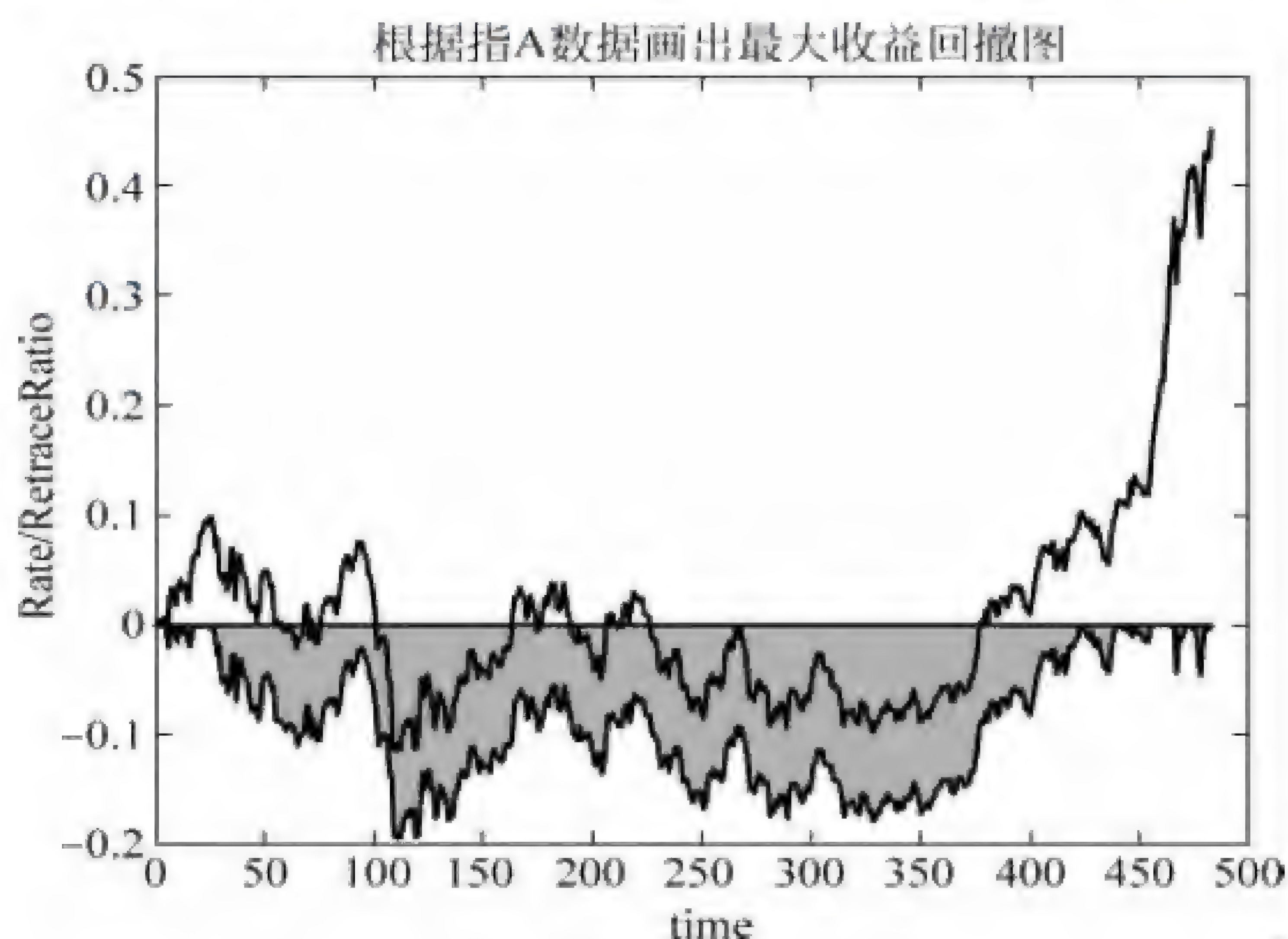


图 18-5　最大收益回撤图

18.4　固定收益证券的久期和凸度计算

固定收益证券,是指持券人可以在特定的时间内取得固定的收益并预先知道取得收益的数量和时间,如固定利率债券、优先股股票等。固定收益证券(fixed-income instrument)也称为债务证券,承诺在将来支付固定的现金数量。

我国市场上的固定收益类产品主要有国债、中央银行票据、企业债、结构化产品和可转换债券。从存量来看国债和央行票据构成了我国固定收益类证券的主体,可转债、结构化产品及无担保企业债也正在快速地发展。

久期是债券价格与利率关系的一阶导数,凸性是债券价格对利率的二阶导数。当两个债券的久期相同时,它们的风险不一定相同,因为它们的凸性可能是不同的。

在收益率增加相同单位时,凸性大的债券价格减少幅度较小;在收益率减少相同单位时,凸性大的债券价格增加幅度较大。因此,在久期相同的情况下,凸性大的债券其风险较小。数学上讲,凸性是债券价格对到期收益率二次微分,再除以债券价格,或者说是二阶导数。

现实中的应用是若预测收益率将下降,对于久期相同的债券,选择凸性较大的品种较为有利,反之则反。

一张 T 年期债券,t 时刻的现金支付为 $C_t(1\leqslant t\leqslant T)$,与债券的风险程度相适应的收益率为 y,则债券的价格为

$$P = \sum_{t=1}^{T} \frac{C_t}{(1+y)^t}$$

其久期为

$$D = \sum_{t=1}^{T} t \left[\frac{\frac{C_t}{(1+y)^t}}{P} \right]$$

其中，C_t 表示 t 时的现金流，y 为到期收益率（每期），P 为债券的现价，T 为到期前的时期数，t 为收到现金流的时期。

凸度实际上是价格-收益曲线斜率的变化率。

付息周期数为 n，周期收益率为 y 的债券的凸度计算公式如下：

$$凸度 = \frac{1}{P(1+y)^2} \sum_{t=1}^{n} \frac{t(t+1)C_t}{(1+y)^t}$$

其中，C_t为 t 时刻的现金支付。

【例 18-7】 已知三只国债的相关信息如表 18-2 所示，编写 MATLAB 代码计算国定收益证券的久期和凸度。

表 18-2　三只国债的相关信息

证券编号	固定收益证券	到期收益率	票面利率	结算日	到期日	计息方式
150008	16 附息国债 08	3.0450	3.5400	16-Apr-15	16-Apr-22	每年付息一次
150027	16 附息国债 27	3.0600	3.0500	22-Oct-15	22-Oct-22	每年付息一次
150018	16 附息国债 18	2.9050	3.1400	8-Sep 15	8-Sep-20	每年付息一次

解：首先计算三种债券的价格及计算日的利息。

```
clear all
clc
% 计算 16 附息国债 08 的价格和结算日利息
Yield = [0.03045];
CouponRate = 0.0354;
Settle = '16 - Apr - 2015';
Maturity = '16 - Apr - 2022';
Period = 1;
Basis = 0;
[Price08, AccruedInt08] = bndprice(Yield, CouponRate, Settle, Maturity, Period, Basis);
disp('16 附息国债 08 的价格');
Price08
disp('16 附息国债 08 的结算日利息');
AccruedInt08

% 计算 17 附息国债 27 的价格和结算日利息
Yield = [0.036];
CouponRate = 0.0305;
Settle = '22 - Oct - 2015';
Maturity = '22 - Oct - 22';
Period = 1;
Basis = 0;
```

```
[Price27,AccruedInt27] = bndprice(Yield,CouponRate,Settle,Maturity,Period,Basis);
disp('16 附息国债 27 的价格');
Price27
disp('16 附息国债 27 的结算日利息');
AccruedInt27

%计算 16 附息国债 18 的价格和结算日利息
Yield = [0.02905];
CouponRate = 0.0314;
Settle = '8 - Sep - 2015';
Maturity = '8 - Sep - 2020';
Period = 1;
Basis = 0;
[Price18,AccruedInt18] = bndprice(Yield,CouponRate,Settle,Maturity,Period,Basis)
disp('16 附息国债 18 的价格');
Price18
disp('16 附息国债 18 的结算日利息');
AccruedInt18
```

运行后,得到结果为

```
16 附息国债 08 的价格
Price08 =
  102.9320

16 附息国债 08 的结算日利息
AccruedInt08 =
     0

16 附息国债 27 的价格
Price27 =
   96.4564

16 附息国债 27 的结算日利息
AccruedInt27 =
     0
Price18 =
  100.9817
AccruedInt18 =
     0

16 附息国债 18 的价格
Price18 =
  100.9817

16 附息国债 18 的结算日利息
AccruedInt18 =
```

再根据债券价格计算三种债券久期

```
clear all
clc
%16 附息国债 08 的久期计算
Price = [102.9320];
CouponRate = 0.0354;
Settle = '16 - Apr - 2015';
Maturity = '16 - Apr - 2022';
Period = 1;
Basis = 0;
[ModDuration08, YearDuration08, PerDuration08 ] = bnddurp ( Price, CouponRate, Settle,
Maturity, Period, Basis)

%16 附息国债 27 的久期计算
Price = [96.4564];
CouponRate = 0.0305;
Settle = '22 - Oct - 2015';
Maturity = '22 - Oct - 22';
Period = 1;
Basis = 0;
[ModDuration27, YearDuration27, PerDuration27 ] = bnddurp ( Price, CouponRate, Settle,
Maturity, Period, Basis)

%16 附息国债 18 的久期计算
Price = [100.9817];
CouponRate = 0.0314;
Settle = '8 - Sep - 2015';
Maturity = '8 - Sep - 2020';
Period = 1;
Basis = 0;
[ModDuration18, YearDuration18, PerDuration18 ] = bnddurp ( Price, CouponRate, Settle,
Maturity, Period, Basis)
```

运行后得到结果为

```
ModDuration08 =
    6.2380
YearDuration08 =
    6.3330
PerDuration08 =
   12.6660
ModDuration27 =
    6.2823
YearDuration27 =
    6.3954
PerDuration27 =
   12.7908
ModDuration18 =
    4.6390
YearDuration18 =
```

```
    4.7064
PerDuration18 =
    9.4127
```

再根据债券收益率计算久期

```
%根据债券收益率计算久期
clear all
clc
%16 附息国债 08 的久期计算
Yield = [0.03045];
CouponRate = 0.0354;
Settle = '16 - Apr - 2015';
Maturity = '16 - Apr - 2022';
Period = 1;
Basis = 0;
[ModDuration08, YearDuration08, PerDuration08 ] = bnddurp ( Yield, CouponRate, Settle,
Maturity, Period, Basis)

%16 附息国债 27 的久期计算
Yield = [0.036];
CouponRate = 0.0305;
Settle = '22 - Oct - 2015';
Maturity = '22 - Oct - 22';
Period = 1;
Basis = 0;
[ModDuration27, YearDuration27, PerDuration27 ] = bnddurp ( Yield, CouponRate, Settle,
Maturity, Period, Basis)

%16 附息国债 18 的久期计算
Yield = [0.02905];
CouponRate = 0.0314;
Settle = '8 - Sep - 2015';
Maturity = '8 - Sep - 2020';
Period = 1;
Basis = 0;
[ModDuration18, YearDuration18, PerDuration18 ] = bnddurp ( Yield, CouponRate, Settle,
Maturity, Period, Basis)
```

运行后得到结果为

```
ModDuration08 =
    0.0931
YearDuration08 =
    1.0086
PerDuration08 =
    2.0172
ModDuration27 =
    0.1093
```

```
YearDuration27 =
    1.0118
PerDuration27 =
    2.0236
ModDuration18 =
    0.0966
YearDuration18 =
    1.0093
PerDuration18 =
    2.0185
```

其次，根据价格计算凸度

```
clear all
clc
%16 附息国债 08 的凸度计算
Price = [102.9320];
CouponRate = 0.0354;
Settle = '16 - Apr - 2015';
Maturity = '16 - Apr - 2022';
Period = 1;
Basis = 0;
[YearConvexity08,PerConvexity08] = bndconvp(Price, CouponRate, Settle, Maturity, Period,
Basis)

%16 附息国债 27 的凸度计算
Price = [96.4564];
CouponRate = 0.0305;
Settle = '22 - Oct - 2015';
Maturity = '22 - Oct - 22';
Period = 1;
Basis = 0;
[YearConvexity27,PerConvexity27] = bndconvp(Price, CouponRate, Settle, Maturity, Period,
Basis)

%16 附息国债 18 的凸度计算
Price = [100.9817];
CouponRate = 0.0314;
Settle = '8 - Sep - 2015';
Maturity = '8 - Sep - 2020';
Period = 1;
Basis = 0;
[YearConvexity18,PerConvexity18] = bndconvp(Price, CouponRate, Settle, Maturity, Period,
Basis)
```

运行后，得到结果为

```
YearConvexity08 =
   44.4010
PerConvexity08 =
  177.6039
YearConvexity27 =
   44.7740
PerConvexity27 =
  179.0958
YearConvexity18 =
   24.5867
PerConvexity18 =
   98.3467
```

最后，根据收益率计算凸度。

```
clear all
clc
%16 附息国债 08 的凸度计算
Yield = [0.03045];
CouponRate = 0.0354;
Settle = '16 - Apr - 2015';
Maturity = '16 - Apr - 2022';
Period = 1;
Basis = 0;
[YearConvexity08,PerConvexity08] = bndconvp(Yield, CouponRate, Settle, Maturity, Period,
Basis)

%16 附息国债 27 的凸度计算
Yield = [0.036];
CouponRate = 0.0305;
Settle = '22 - Oct - 2015';
Maturity = '22 - Oct - 22';
Period = 1;
Basis = 0;
[YearConvexity27,PerConvexity27] = bndconvy(Yield, CouponRate, Settle, Maturity, Period,
Basis)

%16 附息国债 18 的凸度计算
Yield = [0.02905];
CouponRate = 0.0314;
Settle = '8 - Sep - 2015';
Maturity = '8 - Sep - 2020';
Period = 1;
Basis = 0;
[YearConvexity,PerConvexity] = bndconvy(Yield,CouponRate, Settle, Maturity, Period, Basis)
```

运行后，得到结果为

```
YearConvexity08 =
    0.0131
PerConvexity08 =
    0.0522
YearConvexity27 =
   44.7740
PerConvexity27 =
  179.0958
YearConvexity =
   24.5867
PerConvexity =
   98.3467
```

本章小结

作为社会科学中科学性较强的一门学科，经济学本身的发展充满了活力，同时也对社会科学其他学科，特别是管理学、法学、政治学的发展起着重要的推动作用，经济学的研究和应用具有广阔的前景。金融则是经济学应用最为广泛与深入的领域之一。

本章首先简单介绍了期权定价分析的基本概念并举例说明，然后对收益、风险与有效前沿的计算及其MATLAB应用做了介绍，最后对投资组合绩效、久期和凸度计算做了详细说明，并举例分析。

附录　MATLAB基本命令

管理命令和函数	help	在线帮助文件
	doc	装入超文本说明
	what	M、MAT、MEX 文件的目录列表
	type	列出 M 文件
	lookfor	通过 help 条目搜索关键字
	which	定位函数和文件
	Demo	运行演示程序
	Path	控制 MATLAB 的搜索路径
管理变量和工作空间	Who	列出当前变量
	Whos	列出当前变量(长表)
	Load	从磁盘文件中恢复变量
	Save	保存工作空间变量
	Clear	从内存中清除变量和函数
	Pack	整理工作空间内存
	Size	矩阵的尺寸
	Length	向量的长度
	disp	显示矩阵
与文件和操作系统有关的命令	cd	改变当前工作目录
	Dir	目录列表
	Delete	删除文件
	Getenv	获取环境变量值
	!	执行 DOS 操作系统命令
	UNIX	执行 UNIX 操作系统命令并返回结果
	Diary	保存 MATLAB 任务
控制命令窗口	Cedit	设置命令行编辑
	Clc	清命令窗口
	Home	光标置左上角
	Format	设置输出格式
	Echo	底稿文件内使用的回显命令
	more	在命令窗口中控制分页输出
启动和退出	Quit	退出 MATLAB
	Startup	引用 MATLAB 时所执行的 M 文件
	Matlabrc	主启动 M 文件

续表

指数函数	exp	E 为底指数
	log	自然对数
	log10	10 为底的对数
	log2	2 为底的对数
	pow2	2 的幂
	sqrt	平方根
圆整函数和求余函数	ceil	向+∞圆整
	fix	向 0 圆整
	floor	向-∞圆整
	rem	求余数
	round	向靠近整数圆整
	sign	符号函数
矩阵变换函数	fiplr	矩阵左右翻转
	fipud	矩阵上下翻转
	fipdim	矩阵特定维翻转
	Rot90	矩阵反时针 90 翻转
	diag	产生或提取对角阵
	tril	产生下三角
	triu	产生上三角
	det	行列式的计算
其他函数	min	最小值
	mean	平均值
	std	标准差
	sort	排序
	norm	欧氏长度
	max	最大值
	median	中位数
	diff	相邻元素的差
	length	个数
	sum	总和
三角函数	sin	正弦
	sinh	双曲正弦
	asin	反正弦
	asinh	反双曲正弦
	cos	余弦
	cosh	双曲余弦
	acos	反余弦
	acosh	反双曲余弦
	tan	正切
	tanh	双曲正切
	acsch	反双曲余割
	cot	余切
	coth	双曲余切

续表

三角函数	atan	反正切
	atan2	四象限反正切
	atanh	反双曲正切
	sec	正割
	sech	双曲正割
	asec	反正割
	asech	反双曲正割
	csc	余割
	csch	双曲余割
	acsc	反余割
	acot	反余切
	acoth	反双曲余切
复数函数	abs	绝对值
	argle	相角
	conj	复共轭
	image	复数虚部
	real	复数实部
数值函数	fix	朝零方向取整
	floor	朝负无穷大方向取整
	ceil	朝正无穷大方向取整
	round	朝最近的整数取整
	rem	除后余数
	sign	符号函数
操作符和特殊字符	zeros	零矩阵
	ones	全“1”矩阵
	eye	单位矩阵
	rand	均匀分布的随机数矩阵
	n	正态分布的随机数矩阵
	linspace	线性间隔的向量
	logspace	对数间隔的向量
	meshgrid	三维图形的 X 和 Y 数组
	:	规则间隔的向量
特殊变量和常数	ans	当前的答案
	eps	相对浮点精度
	realmax	最大浮点数
	realmin	最小浮点数
	pi	圆周率值 3.1415926535897…
	i,j	虚数单位
	inf	无穷大
	nan	非数值
	flops	浮点运算次数
	nargin	函数输入变量数
	nargout	函数输出变量数

续表

特殊变量和常数	computer	计算机类型
	isieee	采用 ieee 算术标准时，其值为
	why	简明的答案
x - y图形	plot	线性图形
	loglog	对数坐标图形
	semilogx	半对数坐标图形(x 轴)
	polar	极坐标图
	bar	条形图
	stem	离散序列图或杆图
	stairs	阶梯图
	errorbar	误差条图
	semilogy	半对数坐标图形(y 轴)
	fill	绘制二维多边形填充图
	hist	直方图
	rose	角度直方图
	compass	区域图
	feather	箭头图
	fplot	绘图函数
	comet	星点图
图形注释	title	图形标题
	xlabel	x 轴标记
	ylabel	y 轴标记
	text	文本注释
	gtext	用鼠标设置文本
	grid	网格线

参考文献

[1] 孙文瑜，徐成贤，朱德通. 最优化方法[M]. 北京：高等教育出版社 2014.

[2] 陈宝林. 最优化理论与算法[M]. 北京：清华大学出版社，2005.

[3] 解可新，韩健，林友联. 最优化方法(修订版)[M]. 天津：天津大学出版社，2004.

[4] 张威. MATLAB 基础与编程入门[M]. 西安：西安电子科技大学出版社，2004.

[5] 黄友锐. 智能优化算法及其应用[M]. 北京：国防工业出版社，2008.

[6] 曹弋，赵阳. MATLAB 实用教程[M]. 北京：电子工业出版社，2004.

[7] 王志新等. MATLAB 程序设计及其数学建模应用[M]. 北京：科学出版社，2013.

[8] 张光澄. 非线性最优化计算方法[M]. 北京：高等教育出版社，2005. 7.

[9] 邢文训，谢金星. 现代优化计算方法[M]. 北京：清华大学出版社，2000.

[10] 张忠桢. 二次规划——非线性规划与投资组合的算法[M]. 武汉：武汉大学出版社. 2006.

[11] Edward B. Magrab. MATLAB 原理与工程应用[M]. 高会生，李新叶，胡智奇等译. 北京：电子工业出版社，2002.

[12] Franklin G F, Powell J D, Emami-Naeini. Feedback Control of Dynamic Systems. New Jersey: Prentice Hall, 2002.

[13] 史峰等. MATLAB 智能算法 30 个案例分析[M]. 北京：北京航空航天大学出版社，2011.

[14] 王凌. 智能优化算法及其应用[M]. 北京：科学出版社，2004.